Technical Report 03-2008

Learning Qualitative Models by an Autonomous Robot

Ashok Mohan

July 2008

Publisher: Dean Prof. Dr. Kurt Ulrich Witt

University of Applied Sciences Bonn-Rhein-Sieg,
Department of Computer Science

Sankt Augustin, Germany

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

ISSSN 1869-5272

Abstract

This thesis introduces and demonstrates a novel method for learning qualitative models of the world by an autonomous robot. The method makes possible generation of qualitative models that can be used for prediction as well as directing the experiments to improve the model. The qualitative models form the knowledge representation of the robot and consists of qualitative trees and non-deterministic finite automaton. An efficient exploration algorithm that lets the robot collect the most relevant learning samples is also introduced. To demonstrate the use of the methodology, representation and algorithm, two experiments are described. The first experiment is conducted using a mobile robot and a ball, where the robot observes the ball and learns the effect of its actions on the observed attributes of the world. The second experiment is conducted using a mobile robot and five boxes, two non-movable boxes and three movable boxes. The robot experiments actively with the objects and observes the changes in the attributes of the world. The main difference with the two experiments is that the first one tries to learn by observation while the second tries to learn by experimentation. In both experiments the robot learns qualitative models from its actions and observations. Although the primary objective of the robot is to improve itself by being able to predict the outcome of its actions, the models learned were also used at each step of the learning process to direct the experiments so that the model converges to the final model as quickly as possible.

Contents

List of Figures

List of Tables

Listings

Acknowledgments

The work described in this article has been [partially] funded by the European Commission's Sixth Framework Programme under contract no. 029427 as part of the Specific Targeted Research Project XPERO ("Robotic Learning by Experimentation").

Chapter 1

Introduction

The idea of machines (in this work, autonomous robots) that are capable of learning by themselves, without any human intervention is one of the most fundamental goals of AI. Among several paradigms of learning, learning by experimentation demands no teacher, but rather learns autonomously, interacting with the real world. In this work we present a methodology in which an autonomous robot can learn qualitative models by conducting experiments in its environment.

The work described in this report is part of the European collaborative research project XPERO. The objective of XPERO is to enable a robot to learn by performing experiments with objects in the surrounding world. The work in XPERO is, to some extend inspired by the way little children explore and learn about their world by playing and experimenting with objects all day long. For example, they have their own way to learn about various physical laws by experimenting several times with objects by dropping them and observing how they fall. In XPERO we use a similar approach in learning by experimentation. Our robot performs these experiments with the objects in its environment to learn about various laws of physics.

Experimenting is one fundamental path to gaining insights about the world not only for children but also for adults, even scientists. Watching an object and moving around it or pushing it around are all forms of experimentation that can help the robot to understand its world better. By understanding the laws and concepts behind such small and seemingly trivial phenomenon, the robot will become more and more competent in understanding and predicting what is going on in the surrounding world. By building models of similar singular concepts, the robot will be able to predict the result of its actions in the world.

The focus on the work described here is on the question "How can the robot build models of its world, so that it can predict the outcome of its actions effectively"? One method is to build progressive models of the world by performing experiments. These models can then be used by the robot to predict the outcome of its actions. It can also be used in the learning process to direct the experiments so that the robot can learn the models faster.

Consider a scenario where the robot is in an empty space. The robot is equipped with only two actuators which are its left and right wheels. By changing the speeds of these actuators, the robot drives around in this environment. But no matter what changes it makes to the actuators, the sensors will read the same. So the prediction at any step would be that there will not be any change in the observation with respect to the actions of the robot. Now if we introduce an object (a ball or a box) into this environment, the sensors of the robot will detect this. Assuming the robot is equipped with methods to calculate the distances and angles to objects, it will start seeing changes in these observations. It

will notice that the distances and angles change when it changes its actuator values. This will be identified by the robot due to the failure in its prediction. The failure of prediction acts as a source of motivation for the robot in the same way it motivates humans. The biggest motivation for humans is the constant endeavor to improve ourselves. The same motivation is in play here as the robot wants to continually improve itself. In this particular scenario, it translates to the ability to make correct predictions.

Consider a slightly more complex scenario where we add more objects into the world, now the robot has to build its models with respect to all the objects. It measures the distances and angles to all objects. But it is important that the robot has the ability to distinguish between the object it is experimenting from the rest of the objects. With this ability it groups the observations into two categories and build models accordingly.

We describe in detail both scenarios mentioned above and evaluate the results. The first scenario described with the robot and a single object, is described in detail in Chapter. 4 titled learning by observation. In this scenario we do not let the robot perform active experimentation with the object in the world. It is allowed only to observe. The second scenario described with the robot and a set of objects is described in detail in Chapter. 5 titled learning by experimentation. As the name suggests, the robot is allowed to actively experiment with the objects in the world by pushing them around.

There are several ways in which the robot chooses its actions, designs and plans experiments. In order to learn efficiently, the method it uses to explore its environment is very important. The robot is not aware of the co-ordinates of the world. So it cannot randomly sample the entire x-y plane. It is clear that such a sampling is not a natural method to learn. Intelligent beings like us do not just randomly sample in order to learn a new model. Motivated by this, we propose an exploration algorithm for autonomous robot learning. The exploration algorithm tries to sample the most significant sections of the world using the available information(incomplete qualitative models learned). Knowledge representation is another very important factor in learning. Representations which uses a lot of quantitative(numeric) information have by far proven to be abstruse. The work described here makes use of qualitative models for representation of knowledge. Qualitative models are easier to learn and sufficient to design and plan the experiments. They reduce the complexity of numerical models considerably and also enable humans to easily understand what the robot has learned.

Chapter 2

Background

This body of research builds upon work from various fields which include but are not limited to robotics, AI, psychology and cognitive sciences. The work described here primarily deals with qualitative reasoning techniques. This is motivated by human or animal behavior. Humans do not use precise mathematical equations for the computation required for our daily routine, at least not at a conscious level. Building upon this we try to abstract as much numerical information as possible, keeping only those (landmarks) which are significant. In this section we describe the main tools and techniques that we use in this work.

2.1 Qualitative Reasoning

Qualitative reasoning [Weld and de Kleer, 1990] is an area of AI which deals with the creation of representation and reasoning of the world with incomplete knowledge [Kuipers, 1994]. It is motivated by various observations made in the world around us. For example people make very useful conclusions about the world that surrounds them without precise mathematical equations. We manage to comprehend well about what is happening around us and how we can affect it with far less and imprecise data than what is used in usual numerical methods. Enabling robots to operate in unconstrained environments involves a lot of incomplete knowledge as well as an abundance of numeric data. Qualitative models [Kuipers, 1993b] has been proven to be extremely efficient in representing incomplete knowledge. So it is evident that a qualitative description is enough to capture the important distinctions while ignoring the insignificant quantitative details of the world. It is for this very reason that we believe qualitative reasoning will provide the answers to many problems in robotics.

Qualitative reasoning has typically focused on scientific and engineering domains, hence its other name, qualitative physics [Forbus, 1988, de Kleer and Brown, 1984, Collins and Forbus, 1987]. Qualitative physics began with de Kleers investigation on how qualitative and quantitative knowledge interacted in solving a subset of simple textbook mechanics problems [de Kleer, 1990]. Qualitative mechanics is also a similar field that analyses the motion as well as the geometric interactions of physical objects. Nielsen [Nielsen, 1988, Forbus et al., 1991] proposes a theory for analysis of rigid body mechanisms which form a large subset of qualitative mechanics problems. It describes the complete working of several mechanical clocks qualitatively.

All qualitative representations provide notations for describing and reasoning about continuous properties of the world. Two main aspects to consider when choosing a representation are resolution and composition. [Forbus, 1996] Resolution defines the level of detail in a representation whereas composition defines the ability to combine representations for different aspects of the system into a whole. Resolution provides us a way to determine how little information suffices to draw useful conclusions about the system. Composition-

ality provides a way to formalize the modeling process.So both are important aspects to consider when choosing a representation.
Having understood the basic description of qualitative reasoning we will now see some basic terminology used in qualitative reasoning.

Qualitative state A qualitative state consists of a set of propositions that characterize a qualitatively distinct behavior of a system. Humans are able to reason fluently about motion through space [Forbus, 1983]. For example consider the case of a bouncing ball. The qualitative state of the ball when it is dropping to the ground would include information about what physical processes are occurring (for e.g. displacement, acceleration due to gravity) and how the parameters in question are changing (for e.g. its position is reducing and its velocity is increasing with respect to the ground). We can consider one single qualitative state as an abstraction of an infinite number of quantitative states: Although the position and velocity of the ball are changing at each instant while it is falling, all the qualitative state of its motion is unchanged until the ball collides with the ground. Once the ball collides with the ground it bounces off it. This triggers a change in the qualitative state. Now unlike the case where the ball was dropping, the displacement is increasing and the velocity decreasing with respect to the ground.

Qualitative transitions In the case of the bouncing ball, the collision with the ground itself can be said to be a qualitative state. Now we have 3 states in consideration. One where the ball is continuously falling and another when it collides with the ground. The last state would be the ball bouncing up until it stops in mid air to fall back again. This falling, colliding and bouncing can be represented via transitions between the states. One transition will define the change of the state from falling to colliding and another would define the change from colliding to bouncing.

Qualitative behaviors The whole process of the ball falling, colliding and bouncing off the ground is termed as a qualitative behavior [Kuipers, 1984] . More precisely, a sequence of qualitative states occurring over a span of time is called a behavior. They can be either purely qualitative, purely quantitative or both. In case of quantitative parameters, it is same as the notion of trajectory in a state-space model. But if qualitative parameters are used one behavior can represent a set of trajectories through the state-space.
Consider the scenario when there is some obstacle on the way of the ball. A possible behavior is that the ball will collide with that object and deviate significantly from the normal behavior. In general, a qualitative state can have transitions to multiple states, indicating ambiguity in the qualitative representations. This is due to the fact that the model is based on incomplete knowledge. Most of the time, this does not pose a threat as there is nothing better we can do with this incomplete knowledge.

Envisionments A set of qualitative states together with their transitions are called as envisionments [de Kleer, 1990]. In a correct envisionment, every possible behavior of the system will match with some path of the envisionment. But every path in the envisionment may not correspond to a possible behavior. This proves to be a challenge as every envisionment will yield multiple behaviors some of which may not be even achievable. This results in further complexity as we have to remove these impossible behaviors and it may not be trivial in most cases. Envisionments may be augmented with actions to be used in autonomous robots. Action-augmented envisionments [Forbus, 1989] incorporates both the effects of an agent's action and the changes that will occur in the physical world whether or not the agent performs an action. In this work we introduce a new qualitative envisionment using a non-deterministic finite automaton.

A detailed study on the various methodolgies, algorithms and tools related to qualitative reasoning can be found in Section. 3. Now that we have an understanding of the basic terminology used in qualitative reasoning, we will see the other tools and techniques used in this work.

2.2 Parametric Padé

Algorithm Padé (not Parametric) [Zabkar et al., 2007], is in fact a set of algorithms that can estimate partial derivatives from a sampled continuous function. There are various versions of the algorithm in Padé. The basic version called First Triangle is based on splitting the attribute space into regions defined by Delaunay triangulation. The method is simple but performs poorly in the presence of noise. There are several modifications of the First Triangle method. They are called as Star Regression, Triangle's Path and Tube Regression. Star Regression is based on the First triangle method. It assumes the functions linearity across the entire star (a topological term for the set of triangles surrounding a point) instead of just a single triangle. This greatly improves its resistance to noise. Triangle's Path method copes with even more noise by smoothing the function further. The star is not simply widened but instead follow the triangles in the direction in which the derivative is computed. Tube Regression is an approximation of the Triangle's Path method. Instead of computing the triangulation it considers a certain number of examples nearest to the axis in the direction in which we compute the derivative.
Padé, discovers monotonic relations in static domains. It does so by computing partial derivatives from numerical data and can be used together with an appropriate machine learning algorithm, e.g. decision trees, to build a qualitative model. However, it is quite limited in the diversity of the domain types it can handle. For example, it can not handle a temporal data set well. Parametric Padé, abbreviated pPadé was created to remove this constraint. The parameter in pPadé is time which allows pPadé to learn in dynamic domains. pPadé also works with other parameterizations than time. Time as just an example.
Although time is used as part of the learning attribute set, pPadé does not consider it as an attribute, but rather a parameter. Hence the name Parametric Padé. The temporal dimension is in a way hidden. For example, consider the well known parametric equations $x(t) = \cos(t)$,$y(t) = \sin(t)$ represent a unit circle for $t \in [0, 2\pi]$. We observe the circle in xy-plane, where parameter t remains hidden.
pPadé is a very versatile algorithm to determine derivatives w.r.t time, but it lacks the ability to handle the special problems that arises in experimentation with objects. In our work, we could use pPadé directly without any modifications for learning by observation to compute the derivatives of the features or attributes used for learning w.r.t time. But it proved insufficient for learning by experimentation. In learning by experimentation the object under experimentation must be distinguished from the other objects. In order to achieve this we added a new parameter in the learning sample called *object*. This parameter is similar to the *time* parameter as it also acts as a parameter, but it also acts as an attribute for learning. Simple pPadé does not have any notion of objects. Extending it was necessary to use it in a robotic domain. Robotic domains usually consists of objects in the world. pPadé was extended to incorporate this notion of objects. The details on how it was extended is described in detail in Section. 5.1.2.

2.3 Orange

Orange [Demsar and Zupan,] is a library that includes a large variety of machine learning and data mining algorithms. It is also a scriptable environment for fast prototyping of new algorithms. It consists mostly of a collection of Python-based modules that sit over a core

C++ library and implement some functionality for which execution time is not crucial. Orange also offers a nice set of graphical widgets which in turn use methods from the core library and Orange modules. The widgets primarily work on signal-based communication and can be easily assembled together into an application by a visual programming tool called Orange Canvas [Zupan and Demsar, 2004]. In short Orange is a framework for machine learning and data mining.

In this work we mainly use the scripting functionality of Orange. Scripting is made available by the python interfaces to Orange. Python is a modern scripting language which is easy to use due to its liberal syntax restrictions. Python has an added advantage of a large community support with an extensive set of open libraries. Due to its inherent simplicity and ease of use, this entire research work has been implemented in python. Although Orange has an extensive set of python interfaces for various machine learning algorithms as well as data input and manipulation, we use only a handful of those in this work. The main functionalities we use are listed below.

2.3.1 Data Representation

The learning samples in this work was constructed in a manner which is understood by the machine learning tools provided by Orange. In Orange the learning samples are represented in the form of an *ExampleTable*. An *ExampleTable* consists of *Example*s of a particular *Domain*. The *Domain* in turn consists of *Variable*s. An *Example* consists of *Value*s for each of the *Variable*s in the *Domain*. In this section we briefly describe what each of these terms mean and how it was used in this work. A portion of the *ExampleTable* with the different parts used for learning by observation is shown in Figure. 2.1

ExampleTable *Example*s are usually stored in a table called *orange.ExampleTable*. In Python it is simply perceived as a list and this is what it basically is: an ordered sequence of examples, supporting the usual Python procedures for lists, including the more advanced operations such as slicing and sorting. In the learning by observation experiment, the entire learning data is stored in a single *ExampleTable*.

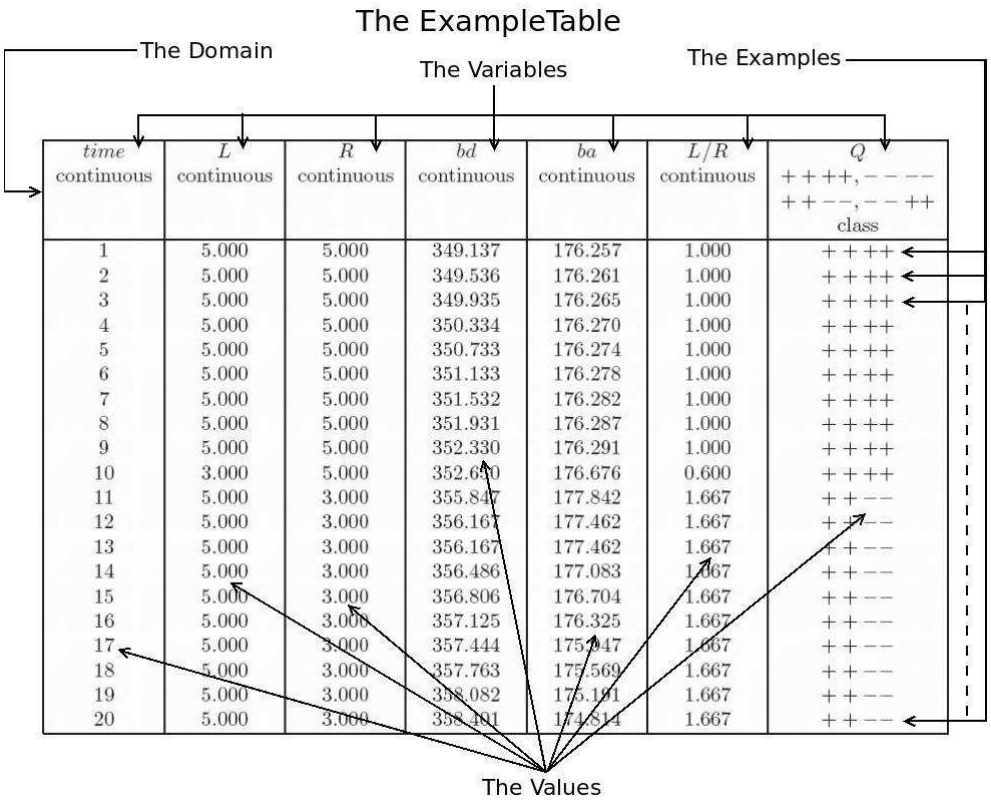
Example *orange.Example* holds examples - a list of attribute values, together with some auxiliary data. Each *Example* corresponds to some *Domain* and therefore the number of attributes ("list elements") and their types are always as prescribed. In the learning by observation experiment, each learning sample is made into an *Example*. For eg:

[5.000 5.000 349.137 176.257 1.000 ++++]

is an *Example* instance from the learning data.

Domain Domain descriptor (*orange.Domain*) is a list of variables. Each example (*orange.Example*) is associated with a domain descriptor, and so are example tables. In the learning by observation experiment, each learning sample is made according to a *Domain*. For eg:

[orange.FloatVariable("L"), orange.FloatVariable("R"), orange.FloatVariable("bd"), orange.FloatVariable("ba"), orange.FloatVariable("L/R")], orange.EnumVariable("Q", values=[++++, − − −−, + − +−, − + −+]



time continuous	*L* continuous	*R* continuous	*bd* continuous	*ba* continuous	*L/R* continuous	*Q* ++++,−−−− ++−−,−−++ class
1	5.000	5.000	349.137	176.257	1.000	++++
2	5.000	5.000	349.536	176.261	1.000	++++
3	5.000	5.000	349.935	176.265	1.000	++++
4	5.000	5.000	350.334	176.270	1.000	++++
5	5.000	5.000	350.733	176.274	1.000	++++
6	5.000	5.000	351.133	176.278	1.000	++++
7	5.000	5.000	351.532	176.282	1.000	++++
8	5.000	5.000	351.931	176.287	1.000	++++
9	5.000	5.000	352.330	176.291	1.000	++++
10	3.000	5.000	352.650	176.676	0.600	++++
11	5.000	3.000	355.847	177.842	1.667	++−−
12	5.000	3.000	356.167	177.462	1.667	++−−
13	5.000	3.000	356.167	177.462	1.667	++−−
14	5.000	3.000	356.486	177.083	1.667	++−−
15	5.000	3.000	356.806	176.704	1.667	++−−
16	5.000	3.000	357.125	176.325	1.667	++−−
17	5.000	3.000	357.444	175.947	1.667	++−−
18	5.000	3.000	357.763	175.569	1.667	++−−
19	5.000	3.000	358.082	175.191	1.667	++−−
20	5.000	3.000	358.401	174.814	1.667	++−−

Figure 2.1: A sub set of the *ExampleTable* from the learning by observation experiment.

is the *Domain* used to construct the examples for learning.

Variable Attribute descriptors are stored in objects derived from type *orange.Variable*. Their role is to identify the attributes. Two attributes in Orange are same, if they have the same descriptor, not the same name. Besides, descriptors store symbolic names for attributes and their symbolic values. In the learning by observation experiment, the *Domain* is constructed using *Variable*s. For eg:

orange.FloatVariable("L")

orange.FloatVariable("R")

orange.FloatVariable("bd")

orange.FloatVariable("ba")

orange.FloatVariable("L/R")

orange.EnumVariable("Q", values=[++++, −−−−, +−+−, −+−+]

are the *Variable*s used to construct the *Domain*. In this work we use two types of variables. *FloatVariable* and *EnumVariable*. as the name suggests, *FloatVariable*s store float values which are continuous. On the other hand *EnumVariable*s store discrete values. It is required to specify all the values that an *EnumVariable* can hold when the domain is defined. This creates additional complexity for our experiments as the class

values are not available at the beginning of the experiment. So we need to recreate the domain everytime we find a new class value.

Value *orange.Value* contains a value of an attribute. *Value* can be discrete, continuous or of some other type, like discrete or continuous distribution, or a string. In the learning by observation experiment, each learning sample has a list of *Value*s which form an *Example* according to the *Domain*. For eg: In the *Example*

[5.000 5.000 349.137 176.257 1.000 + + ++]

each item in the list is a *Value*.

2.3.2 Machine Learning

Orange provides various machine learning and data mining tools like Association rules, C45 Learner, Classification and regression trees, Clustering, k-nearest neighbors, Linear classifier, Logistic regression, Lookup classifiers, Naive Bayes classifier, Random classifier, Rule learning (CN2 etc.), Support vector machines and many more. Among these the tool widely used in our methodology is the Classification tree. We also used Orange Statistics for Predictors which provides various measures of quality for classification and regression.

Classification and Regression Trees In our work we used the *orngTree.TreeLearner* class for classification tree learning. TreeLearner is a class that has built in methods for classifying a set of examples. Upon initialization if it is given a set of examples, it returns a *TreeClassifier*. A *TreeClassifier* is an object that classifies examples according to a tree stored in a field tree. Classification trees are represented as a tree-like hierarchy of *TreeNode* classes. *TreeNode* stores information about the learning examples used for the creation of a node, a branch selector, a list of branches (if the node is not a leaf) with their descriptions and strengths, and a classifier.

The learning samples collected by the robot are converted into an Orange *ExampleTable*. This *ExampleTable* is then used to invoke the *orngTree.TreeLearner* which then return a *TreeClassifier*. Now whenever the robot needs to make a prediction, all it needs to do is to construct an incomplete sample (test data) and pass it to the classifier. The classifier will then classify this test data and return the class value which resulted. This class value in the learning by observation experiment is of the form $++--$ or likewise. From this the robot can deduce that the *bd* will increase and *ba* will decrease if it executes the particular action.

Orange Statistics for Predictors The Orange Statistics for Predictors module contains various measures of quality for classification and regression. The general measures of quality provided are classification accuracy, average probability, Brier's score, information score, confusion matrix, Receiver Operating Characteristic (ROC) analysis, area under ROC curve (AUC) and so on. Among these the most interesting for us is the AUC (Area Under ROC Curve). It returns the area under ROC curve (AUC) given a set of experimental results. The AUC varies between 0 and 1, 1 being the best classifier. We use AUC to determine if a tree created using new learning samples is better than the existing tree. Every time the prediction proves to be wrong, we build a new tree with the all the available data. We then check the AUC of this new tree. If it is same or better than

the old tree, then we update our model with this new tree. Otherwise we keep the old tree.

2.3.3 Simon

Simon is a fast 2D simulator which is a part of Orange. It is a very basic simulator which is especially useful for fast prototyping. It provides various interfaces for creating an environment, moving the robot as well as measuring distances and angles. Although Simon has very limited features, it is entirely written in python making it very easy to add new functionality. Figure. 2.2 shows Simon being used for the learning by observation experiment.

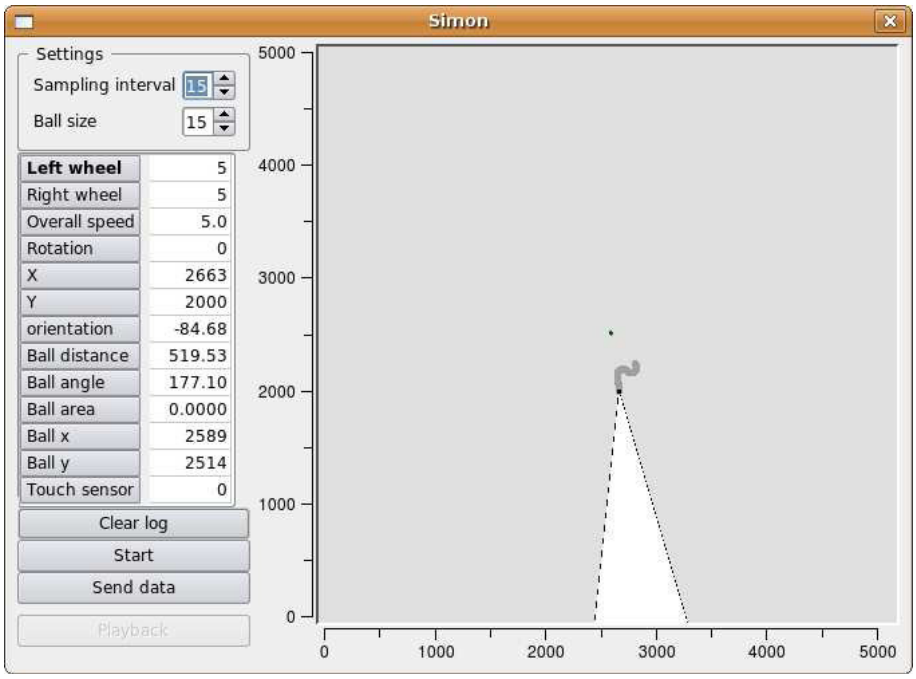


Figure 2.2: Simon running the learning by observation experiment.

As can be seen in the figure, Simon also provides facilities to display the various attributes of the robot as well as the ball. It also shows the path taken by the robot(in grey). Though it is very good for fast prototyping, it is not versatile enough for complex experiments, especially those that involve real world physics. We use Simon for the learning by observation scenario. The learning by experimentation scenario was too complex for Simon as it required real world physics. Hence the learning by experimentation scenario was implemented using Webots instead of Simon.

2.4 Webots

Webots [Webots, , Michel, 2004] is a commercial 3D mobile robot simulation software developed by Cyberbotics Ltd. It is a very versatile and rapid prototyping environment, that allows the user to create environments in 3-Dimensions. It also provides functionality to set and use physics properties such as mass, joints, friction coefficients, etc in the simulation. Objects resembling real world objects can also be added easily with the provided visual interfaces of the software. It also supports different types of mobile robots

with different locomotion schemes (wheeled robots, legged robots, or flying robots). These robots may be equipped with a variety of sensors and actuators such as distance sensors, drive wheels, cameras, servos, touch sensors, grippers, emitters, receivers, etc. It even supports creation of new robots easily using the GUI-tools provided. Environments and robots can be easily created from the GUI of the software. Behaviors for the robots can be programmed in C++, Java or Python. It also provides easy to use interfaces to real mobile robots so that the control programs may be tested on them. Currently it supports robots like Khepera, Hemisson, LEGO Mindstorms, Aibo, etc.

The learning by experimentation scenario of this work has been done entirely using the Webots simulator. The controllers were written in Python. The distances and angles were computed using the *Supervisor* module of Webots. A supervisor is a program which controls a world and its robots. In Webots they are represented as another robot without wheels. It is driven by a separate controller with extended capabilities which may be used to supervise the world. The *Supervisor* module provides methods which can be used to get the position of any object, including the robot in the simulation. We use this position information to calculate the distances and angles.

The distances are calculated using the following simple distance formula $d = \sqrt{(obj\_x_2 - robot\_x_1)^2 + (obj\_y_2 - robot\_x_1)^2}$ for co-ordinate geometry. The calculation of angles are slightly more complex as we need to normalize the angle from the $0 to 2\Pi$ scale to the $-\Pi to \Pi$ scale. The angle is calculated by finding the angle between the orientation of the robot and the line that joins the center of the robot to the center of the object. This is then converted into degrees. The angle is calculated using the formula $(atan2(obj\_y - robot\_y, obj\_x - robot\_x) - robot\_orientation) * 180/\Pi$.

The robot and the supervisor controllers are equipped with emitters and receivers. An emitter is a device which can send data in the form of packets. A receiver receives this data from a queue. The use of emitters and receivers ensure that even though we use the supervisor module to get the positions of the objects, later it can be easily replaced with another system, for example a vision system that computes the distances and angles. When the robot needs to know the distance and angle of any object, it sends a request to the supervisor through the onboard transmitter. The supervisor controller receives this request through the receiver device and processes it. It then transmits the distance and angle of the object to the robot using its emitter device. The robot then uses this information to create learning samples.

Chapter 3

Related work

This section briefly describes the various methodologies and reasoning techniques that has been well established in this area of research. An understanding of the current state of research in qualitative reasoning is essential in the appreciation of this work.

3.1 Qualitative Simulation

Qualitative simulation [Kuipers and Berleant, 1988, Kuipers, 1993a] predicts all possible behaviors of a physical system based on the model of the system specified as a Qualitative Differential Equation (QDE). A QDE is an abstraction of an Ordinary Differential Equation (ODE). It is a qualitative representation paradigm that can successfully represent models of systems with incomplete knowledge. *QSIM* [Kuipers and Berleant, 1988, Kuipers, 1994] is a software/tool that provides the representations and algorithms necessary for qualitative simulation. It is completely based on QDEs. Simulation provides automatic generation of the qualitative behaviors of continuous, real-valued state variables given only a qualitative description of the structure of the mechanism and an initial qualitative state. That is, it predicts the possible behaviors consistent with incomplete knowledge of the structure of physical system. ODEs may seem appropriate for modelling the real world, but ODEs are inherently weak in expressing incomplete knowledge. QDEs are similar to ODEs but are able to express more incomplete knowledge than ODEs. They provides an efficient way to define a qualitative model. The models for QSIM hardly contain any numeric information. This has both advantages and disadvantages. The advantage is that models can be built with very little information. This is especially useful in domains in which there is very little numeric data available. But in domains like robotics where numeric data is available in abundance, this may prove less than fruitful. The models of QSIM abstracts too much information that is available from the sensors of a robot. On one hand this leads to simpler models that can be constructed and used with ease, on the other hand predictions using these models do not include important numeric information that may have been useful.

3.2 Semi-Quantitative Reasoning

Qualitative simulation is powerful in dealing with incomplete knowledge upto ordinal relations of properties of the system. But most of the time we also have some incomplete quantitative information available. This quantitative information may not be enough to perform numerical simulation but may prove very efficient in refining the predictions of a qualitative simulator. Quantitative representations are powerful in terms of its precise formulations but lacks the ability to represent incomplete knowledge. Qualitative repre-

sentations on the other hand is powerful in its ability to express incomplete knowledge but lacks the precise formulations guaranteed by quantitative methods. That is neither quantitative nor qualitative representations can do everything the other can. It is clearly evident that combining both representations would be largely beneficial to deal with those cases which would be impossible using either of these alone.

Q2 [Kuipers and Berleant, 1988] and its successor *Q3* [Berleant, 1991, Berleant and Kuipers, 1992] presents a method for incrementally exploiting incomplete quantitative knowledge, by using it to refine the predictions of a qualitative reasoner. Partial quantitative information is used in the form of intervals. Intervals are intermediate in specificity between numbers and qualitative values. In the absence of this partial quantitative information Q3 performs as a normal qualitative simulator. In case complete quantitative information is available, Q3 performs as a normal numeric or interval simulator. One frequently has quantitative knowledge as well as qualitative. Q3 provides a method to exploit this incomplete quantitative knowledge efficiently by using it to refine the predictions of a qualitative reasoner [Berleant and Kuipers, 1997].
The quantitative information may even be in the form of probability distribution functions. Adding quantitative information to qualitative models clearly helps in defining physical systems more precisely. This increased precision is especially useful to remove the impossible behaviors that usually result from pure qualitative simulation. Addition of this partial quantitative information will also help us to discard many irrelevant models when we are trying to match a stream of observations to multiple models.

QUIN [Bratko and Suc, 2004,Bratko et al., 1992,Suc and Bratko, 2001] is a system that looks for qualitative patterns in numeric data to create qualitative trees. These qualitative trees can be refined to create models of the system. The leaves in these qualitative trees are labelled with what is called as qualitatively constrained functions (QCF). QCFs are analogous to the monotonicity constraints, also known as "multivariate monotonic function constraints" widely used in qualitative reasoning. QUIN takes as input a set of numerical examples and looks for regions in the data space where monotonicity constraints hold. All the identified patterns are represented in the form of a qualitative tree. A qualitative tree is similar to a decision tree in the fact that the internal nodes in a qualitative tree specify conditions that split the attribute space into subspaces. Additionally each leaf specifies a QCF that holds among the input data that fall into that leaf. The work described here also makes use of qualitative trees as we found that it can capture important quantitative information, while qualitatively representing the world.

SQsim [Kay, 1998] is a system which provides a general language for representing and reasoning of models that contain both parametric and functional uncertainty. The models are defined qualitatively adding all available quantitative information to it. SQsim then produces predictions that are consistent with the imprecision of the model. SQsim uses a multi-level representation to keep the precise qualitative information separate from the imprecise quantitative information. This helps in creating unambiguous inferences from the precise qualitative description.

Dynamic envelopes [Kuipers, 1993a] may also be used to exploit quantitative knowledge more by deriving and simulating an external system whose solution is guaranteed to bound all solutions of the SQDE. *NSIM* [Kuipers, 1993a] is a system that can simulate using dynamic envelopes. A model created with QDEs augmented with envelopes for all monotonic functions and numeric ranges for all model variables are called SQDEs (Semi-Quantitative QDEs). The bounds on the SQDE as a function of time are called the dynamic envelopes. Unlike *FuSIM* [Shen and Leitch, 1991] and *Q2*, dynamic envelopes do not produce overly conservative bounds. This is because it uses a simulation time-step determined by qualitative distinctions. The precision of the dynamic envelope depends

only on the precision of the model.

Most of the available semi-quantitative systems except *QUIN* uses QDEs as their qualitative models. Although they perform well for static domains with incomplete knowledge, they are not well suited for a robotics domain which is completely dynamic with abundant numeric information. [Mohan, 2008,Mohan et al., 2007] describes the application of QDEs for a similar domain as this work, but with little success. The specification of a QDE is very fragile and hence is only suitable if they are defined by human experts in the domain. Learning them is a very difficult task and research in this direction has not lead to much success.

3.3 System Identification and Automated modeling

System identification takes a set of possible models and a stream of observational data of a physical system, and attempts to identify the part of the model space that best describes the observed system. In traditional approaches, [Ljung, 1986] the model space is specified by a parametrized differential equation, and identification selects numerical values so that the prediction of the model best matches the observations.
Modeling physical systems manually may prove extremely tedious especially if the physical system is not restricted. For the case of an autonomous robot in an unrestricted world, the robot has to interact with infinitely many objects and physical systems. It would be extremely helpful if we have a mechanism for building the models automatically as and when the robot interacts with them. There are various methods available to perform this. One of it is to automatically abduct behaviors [Bradley and Stolle, 1996] from a description of behaviors. This description of behaviors may be created from a stream of observations.
Revising models can be extremely complex due to the fact that you need domain expertise in order to do this. [Daniel J. Clancy and Kay, 1997] presents different techniques for revising models based on the observations. Model revision is also known as model refining. Some work has also been done in selecting the most appropriate model that fits a set of observations. This mostly deals with selecting quantitative models using qualitative reasoning [Capelo et al., 1996].

SQUID [Kay et al., 2000,Kay, 1997] is a system identification method which attempts to match semi-quantitative trends to semi-quantitative behaviors. For this, the space of potential models is defined by semi-quantitative differential equations. It uses SQSim and MSQUID as components. SQUID uses a conservative refinement technique which eliminates only those QDEs from the model space which do not fit the observation data. But it operates as a batch computation over the measurement stream and not as an online prediction-observation system.

QSI [Say and Kuru, 1996] is a qualitative system identification algorithm that can create constraint models of the system based on qualitative behaviors. Say and Kuru proposes an algorithm for qualitative system identification which builds modelfrom incomplete information. Unlike similar systems which composes models starting from a base model or by combining different model fragments, QSI creates the model fragments from scratch. It can be called as a model fragment formulation tool. It takes as input a set of QSIM behaviors of the system to be identified and outputs a QSIM-style QDE which describes the system. QSI arrives at the model fragments by a brute-force search as there is no initial model available to begin with. The best use of QSI would be to prepare initial models of a system which can then be revised based on real world interactions using other approaches of learning.

LYQUID [Gerceker and Say, 2006] Gerceker and Say proposes a new algorithm intended for the automatic extraction of qualitative models from observed numerical data. Samples from observation data are used to make polynomial approximations to the real world functions hidden behind the observations. This method of approximation is also very robust in terms of noise because approximations are generated from a group of samples, which softens the effect of noise on individual samples. It has been proven to perform well even when there are parts of the observation stream missing. In this case LYQUID fills in the missing parts of the data by extending neighboring approximations over to the unobserved time interval. But when more than one polynomial is fitted over the samples, LYQUID is not so successful in computing monotonic functions and landmark value pairs.

MISQ [Richards et al., 1992] is a system for constructing models purely based on behavior information. Models are generated based on the specificality of the input behaviors. If the system gets the complete behavior information as input then the resulting model is unique and guaranteed to reproduce the input behaviors. But if the input to the system is incomplete information, it will still generate a model that will reproduce the input behaviors but the model may not be unique. MISQ performs model generation in 3 steps. In the first step, if quantitative data is available, it is converted into qualitative behaviors. In the second step individual constraints are generated and tested. Constraints are created consistent with the input. In the third step, models (QDEs) are constructed from the set of constraints generated in the second step. One of the drawbacks of MISQ and also many other similar systems is that it generates many obvious constraints. More filters need to be designed to eliminate these constraints.

GENMODEL [Coiera, 1989] is another system that constructs maximally constrained qualitative models given completely specified qualitative behaviors. MISQ uses the same method as used by GENMODEL to generate the set of constraints but it generates fewer than GENMODEL. GENMODEL does not process quantitative behaviors, work with incomplete information, or perform dimensional analysis. The algorithm proceeds in mainly four steps. It first constructs landmark lists for each function by analyzing the example behavior. Given this set of functions and example behavior it then generates the initial search space consisting of a large but finite number of constraints. Each subsequent example is then used to filter the generated constraints. Any constraint that has no example to support it is deleted. Finally when all the examples have been processed, the remaining constraints form the model. Even in this set, there may be redundant constraints like M+(a,b) and M+(b,a). GENMODEL identifies these redundant constraints and removes them in the final step of the algorithm. The main problem GENMODEL faces is in the challenge to remove some constraints because of insufficient examples.

GOLEM [Muggleton and Feng, 1990, Bratko et al., 1992] is another system that can abduct qualitative models using a logic based approach to machine learning. It uses hand generated negative information which are examples of behaviors the system does not produce. GOLEM uses and follows the QSIM formalism. GOLEM is actually a simplified and more efficient version of CIGOL [Muggleton and Buntine, 1988]. The main advantage of GOLEM over GENMODEL was that it could introduce new variables. This was proven by the famous U-Tube experiment. On the other hand GOLEM requires a large amount of background knowledge to arrive at a reasonable model from few example behaviors.

QMN [Dzeroski and Todorovski, 1995] generates qualitative models from behaviors which are numeric. It does not transform the numeric behaviors to qualitative behaviors. *QMN* achieves this by fitting qualitative constraints or rather QDE constraints directly on to the numeric data. It takes as input a behavior trace of a dynamical system. In addition to this it requires four other parameters. The order of the highest derivative of the dynamical system, the maximum depth of new variables introduced by combining

old variables, and two tolerance levels used for testing the qualitative constraints. The tolerance level also specifies the amount of noise the system can handle. Unlike *MISQ*, *QMN* translates the numerical trace directly into models without transforming them to qualitative behaviors. The results of the experiments conducted with the classic U-Tube model is promising. Nevertheless, the need to specify the order of the highest derivative of the system makes it much less versatile compared to MISQ.

PRET [Bradley and Stolle, 1996] is program that can construct models from hypotheses, observations and specifications. It constructs ODE models from user entered information. Envelopes to monotonic functions may be derived from observations [Ungar, 1993, Kay et al., 2000]. These envelopes can be used to refine models created so that they do not predict impossible behaviors. Machine learning techniques may also be used to automate the task of model building.*PHINEAS* [Falkenhainer, 1990] is a system that seeks the best match between an observation to be explained and the understood phenomena. It can explain a system from existing theories as well as form theories and revise them if necessary. The strength of PHINEAS is its ability to generalize a theory in response to an unanticipated observation.

MIMIC [Dvorak and Kuipers, 1989, Dvorak and Kuipers, 1991, Dvorak, 1992] is a model-based method for monitoring dynamic systems in which the condition of the physical system is represented in a dynamic qualitative model. As the name clearly implies, the system tries to mimic the condition of the physical system in the model. The system works in mainly four steps. First it generates a fault hypothesis via a decision tree of the observations of the system. The second step is then to actually build a model based on the fault hypothesis generated in step one. The third steps proceeds to use QSIM to simulate each of the new fault models starting from the initial state of the model created by initializing the model with the observations that first evoked its creation. The fourth and final step is then to match by computing the similarity between the observations and the state of the model. Matches are retained as plausible reflections of the model and mismatches are discarded.

QPC and SQPC Farquhar presents a Qualitative Physics Compiler (QPC) [Crawford et al., 1990, Farquhar, 1993] and also a semi-quantitative physics compiler [Farquhar, 1994] that can automatically build and simulate qualitative models of physical systems. QPC builds on top of the expressiveness of QP Theory and the mathematical advantages of QSIM. QPC takes a domain theory and scenario as input specified in the QPC modeling language. A set of quantified definitions, called model fragments define the domain theory. The model fragment describes some feature of the domain, such as physical laws, processes, or entities. The scenario definition lists the set of entities that are of interest, initial conditions as well as boundary conditions. Rajagopalan [Rajagopalan, 1994] has used QPC successfully for constructing models for geometric and spatial reasoning. The work involves creating models for magnetic fields problem solver. *SQPC* [Farquhar and Brajnik, 1994] on the other hand makes use of available partial quantitative knowledge. It tries to unify compositional modeling techniques with semi-quantitative simulation. SQPC automatically constructs the semi-quantitative models and provides useful predictions with incomplete knowledge. It has been experimented only with a classical example of a water supply control so its capability to deal with more complex tasks in robotics is doubtful.

QuMAS [Mozetic, 1987] learned models of the electrical system of the heart capable of explaining many types of cardiac arrhythmia. It did not use QDE constraints as modelling primitives, but a set of problem-specific logical descriptions used by what would now be recognised as an ILP learning system. *Garp* [UvA, 2005] is a workbench for qualitative reasoning and modelling. It offers an integrated set of tools for building qualitative models. It also facilitates running and inspecting simulations based on those models. The main advantage of Garp compared to other QR systems is that it provides the user with

a rich array of GUI tools which eases the process of creation of models. Nevertheless the simulation techniques used within the workbench are similar to other simulation software like QSIM.

3.4 Qualitative Spatial Reasoning

Spatial reasoning is a process of forming ideas using the spatial relationships between objects. One of the most widely used example of spatial reasoning is geometry. Qualitative spatial reasoning(QSR) [Forbus, 1995] on the other hand uses qualitative descriptions to form ideas. One of the most relevant work in this area is qualitative spatial reasoning about motion. Unlike qualitative dynamics, purely qualitative spatial representations have not proven fruitful. *FROB* [Forbus, 1980, Forbus, 1983] is a system that reasoned about the motion of balls bouncing around. FROB could predict the specific motion of a ball. It could also produce summaries of the eventual fate of the ball. The most interesting thing it could do was to predict if two balls would collide with each other. Some other spatial reasoning work like the *CLOCK* [Forbus et al., 1991, Forbus et al., 1990] project achieved the qualitative simulation of a mechanical clock from first principles.
Surveys in [Cohn et al., 1997, Cohn, 1997, Cohn and Hazarika, 2001] describes the various challenges faced by QSR. One of this is the fact that space is a multidimensional quantity and cannot be adequately represented by single scalar quantities. A limiting factor to this is that quantity spaces do not work in more than one dimension. In most cases spatial reasoning in our everyday interaction with the world around us is driven by qualitative abstractions rather than total quantitative information. QSR addresses many different aspects of space including topology, distance, size orientation and shape. Among others distance and size are of extreme importance to our experiments. A hierarchy of qualitative representations for space has been specified by Kuipers [Kuipers, 1996]. Especially an ontological hierarchy is specified for large-space.
Spatial representation of distance consist mainly of two types. One that describes distance in some absolute scale and the other that provides some form of relative measurement. Most of the traditional QR representations deal with absolute measurements of distance. Representations like `x(>,d)y` [Zimmermann, 1993] saying that x is larger/bigger than y by an amount d are the most commonly found. The relative scale specifies if two objects are equidistant, nearer than and farther than without specifying by how much. Liu [Liu, 1998] defines qualitative distance and orientation based on angles explicitly and formulates a representation for qualitative trigonometry. The framework for representing distance and orientation [Hernandez et al., 1995] qualitatively is of particular interest to us.

3.5 Recent related work

Similar to our approach, [Modayil and Kuipers, 2007] present an algorithm for learning qualitative models from robot's actions and observations, but their qualitative models are in the form of object control laws while we use qualitative trees and envisionment which are much easier for human comprehension. An interesting approach using probability estimates is described in [Hart et al., 2005]. Work by [Barto et al., 2004] in intrinsically motivated learning shows how reusable rules can be learned, but only in a playroom domain with much more data than we require. [Kuipers et al., 2006] describes a methodology that bootstraps knowledge from low-level sensorimotor primitives and then uses this knowledge to navigate in its environment. [Stoytchev, 2005] proposes a novel approach to representing and learning tool affordances by a robot by pushing objects, but with very limited and specific exploratory behaviors. [Mugan and Kuipers, 2008] presents an approach to learn a qualitative state representation that can be applied to reinforcement learning problems. The reinforcement learning problems are created by the agent itself.

They are very simple because the learned abstraction provides a mapping from the continuous input and motor variables to discrete states that aligns with the dynamics of the environment. The algorithm is evaluated using a *robot baby* which tries to hit a block to the left, right or forward. Although the approach as well as the domain is very similar to the work described here, it takes over 340,000 steps to learn the model.

Chapter 4

Learning by observation

4.1 Approach

This section outlines the structure of the qualitative models as well as an exploration algorithm which has the objective of learning qualitative models using the least possible number of learning samples.

Our task is to enable the robot to learn the effect of its actions on the environment without any external intervention. In order to do this we equip it with an exploration algorithm that would learn models of the world using the least number of learning samples. The models learned are qualitative in nature which has an added advantage that the insights learned would be easily comprehensible to humans. The robot is restricted to learn by observation alone. This means it is not allowed to interact with the objects in the world, but only observe them while moving. The learning happens by progressively building qualitative models as and when new information is available. It then uses these unfinished models for directing the experiments so that the model converges to the final model quickly.

The robot's motivation for learning is a part of the built-in algorithm. Since the robot depends on its own model, the robot wants to optimize the model's prediction accuracy. To improve the model in time requires collecting new observation data, particularly data most useful for the improvement of the model. Initially the data collected is insufficient to make correct predictions, so most of it will be used to improve the model. But as the model improves in the prediction accuracy, lesser learning data will used for improving the model as the robot will begin to make more correct predictions. After some time the robot will come up with a model whose predictions are always correct, which indicates that the robot has learned everything about its actions and their effects in the given environment. It can then move on to conduct more complex experiments to learn more about the world.

4.1.1 Experimental domain

Our problem domain consists of a mobile robot and a ball, as shown in Fig. 4.1. The robot observes its distance to the ball (ball distance, henceforth referred to as *bd*) and the angle between its orientation and the line joining the center of the robot and the ball (ball angle, henceforth referred to as *ba*) as shown in Fig. 4.2.

The robot is of differential drive type and moves by setting the speeds of the left and the right wheel (L and R respectively). In our case, L and R are always positive, and the robot was restricted to choose between speeds 4 and 5 only. So the robot can move straight ahead ($L = R = 5$), right ($L = 5, R = 4$) and left ($L = 4, R = 5$), as shown

Figure 4.1: The experimental setup consisting of a robot and a single object, a ball. There is no restrictions made to the area of the experimental setup so that the robot can move around without obstruction.

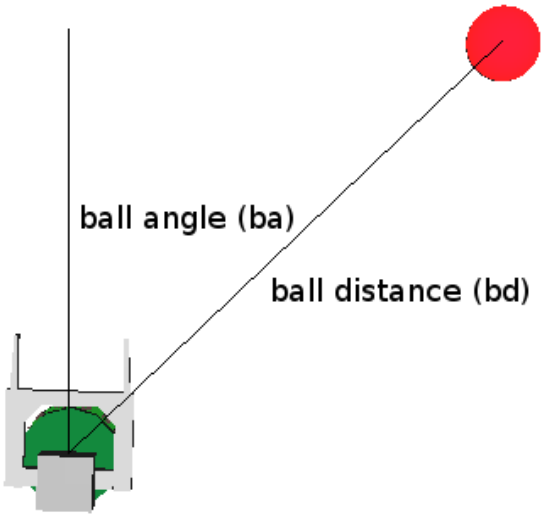


Figure 4.2: The robot, the ball and the observed features. There are only two observed features here, namely *balldistance* (*bd*) and *ballangle* (*ba*) . The *balldistance* is measured as the distance from the center of the robot to the center of the ball. The *ballangle* is measured as the angle between the orientation of the robot and the line joining the center of the robot and the ball.

in Fig. 4.3. A numeric global coordinate system may seem the easiest way to move in a 2D space. But this is in reality counter intuitive since in the real world, the robot will not have the luxury of a global coordinate system. In addition, intelligent beings like us do not actually move using a coordinate system. We use relative information like near, far, towards, away etc in order to navigate. Motivated by this, our robot is not aware of any coordinate system. It is only aware of the actions it performs (L and R) and the observations from the sensors namely *bd* and *ba*.

The overall goal that we want the robot to achieve is that it learns a qualitative model describing the relations between its actions and observations. This model may then be used to make predictions by the robot. Predictions of how the observation attributes will change for changes in the robot's actions. By densely sampling the whole space of above mentioned variables and learning a qualitative tree we obtained an "almost ideal" model of our domain. We did not use this model in any other way than to see for ourselves what the robot should eventually learn. This "almost ideal" qualitative tree for our domain is shown in Fig. 4.4.

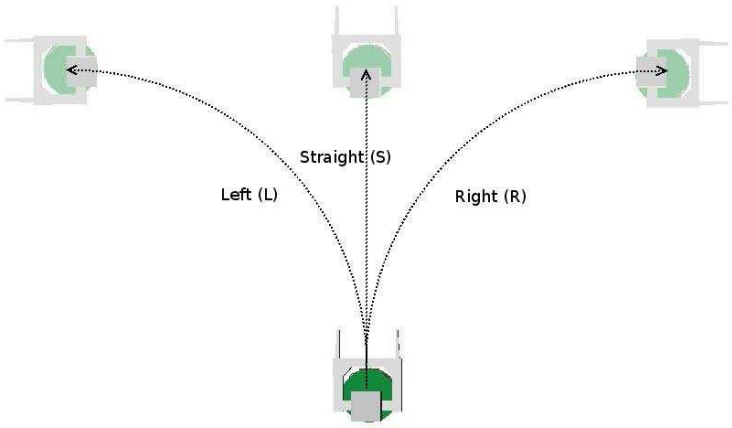


Figure 4.3: The actions of the robot. At any point in time, the robot is allowed to choose from three actions namely left, straight and right. Transcribing this in the robot's term, left would be when $L = 4$ and $R = 5$, straight would be when $L = R = 5$ and right would be when $L = 5$ and $R = 4$. The robot is able to distinguish between these using the pPadé constructed attribute L/R will have values 0.8, 1 and 1.25 for left, straight and right respectively.

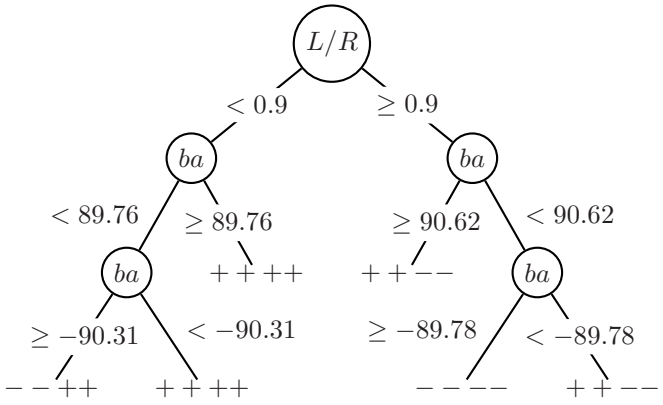


Figure 4.4: The "almost ideal" model of the robot in our domain. This model was achieved by densely sampling the whole space. The only purpose of this model was for a reference for us to understand what the robot should eventually learn.

The ideal model was created using algorithm pPadé along with classification trees to learn qualitative trees. The features we used for constructing this model are the actions L and R, the observed features bd and ba, and a constructed feature L/R. The constructed feature is not a specifically created feature. In another domain with more actions, it would be the ratios between all similar actions. It is constructed by pPadé by the chain rule, dividing the derivatives of each wheel's path w.r.t. time. The attribute L/R describes the qualitative relation between both speeds and can, as we shall see, explain the left and right turns. Using the chain rule for attribute construction is a general principle and is not specifically added to this domain. This ratio gives meaning to the left, right and straight turns of the robot. For left turns L/R will have a value less than 1, for right turns it will have a value greater than 1 and for straight movements it will have a value equal to 1.

The values of the class attribute forms the the leaf nodes (eg: $++--$) of the tree. In our simple example, the robot has two actions (L and R) and two observation variables (ba and bd), so it should learn *bdL=Q(s L)*, *bdR=Q(s R)*, *baL=Q(s L)* and *baR=Q(s R)*,

where sign s is $+$ or $-$ and $L = s\dot{L}, R = s\dot{R}$, where sL and sR are the paths of the left and the right wheel respectively. In these equations, Q stands for qualitative relation as described in Section. 2.2. In the paper, we use a shorter notation, e.g. "$++--$", giving only the signs s in the above mentioned order. So "$++--$" means: $bdL = Q(+L)$, $bdR = Q(+R)$, $baL = Q(-L)$ and $baR = Q(-R)$. In other words, bd is increasing when L and R are increasing (i.e. $L, R > 0$), and ba is decreasing when L and R are increasing. We define the class C of this domain as a 4-tuple of signs as shown in Listing. 4.1.

Listing 4.1: The 4-tuple making the class attribute

C=<bdL , bdR , baL , baR>

Figures 4.5 and 4.6 shows the regions of different values of the class attribute C.

The ideal model indicates a method of feature selection in play here. As you can see the model does not use bd in it. It is clear from this model that bd is not a significant feature as it does not influence the change caused by the robots actions. At first it may seem counter intuitive, but a deeper analysis reveals the case to be true. As shown in Fig. 4.5, when the robot is facing the ball $ba = 0$, a left turn would result in the bd decreasing and ba increasing. This is the same when $ba = -90$as shown in Fig. 4.6. Similarly a right turn would result in bd decreasing and ba decreasing when $ba = 0$ and $ba = 90$. But this behavior remains same no matter what the value of bd is. i.e for $bd = 10cm$ or $bd = 1m$ the effect of the robots actions depend only on the ba at that moment. Fig. 4.5 and Fig. 4.6 shows all the various angles and the corresponding change in the qualitative class value based on the robots actions. Fig. 4.7 and Fig. 4.8 shows a simpler representation that indicates the different class values for the different quadrants of the angles. A more detailed discussion on the inherent feature selection mechanism can be found in Section. 4.2.1.

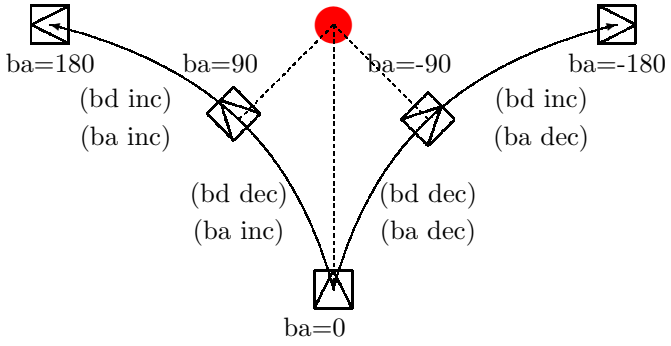


Figure 4.5: The various angles when robot is turning left and right from $ba = 0$. From $ba = 0$ to $ba = 90$ the class value will be $--++$. From $ba = 90$ to $ba = 180$ the class value will be $++++$. From $ba = 0$ to $ba = -90$ the class value will be $----$. From $ba = 90$ to $ba = 180$ the class value will be $++--$.

The experiments were performed using the simulator Simon which is a part of the machine learning framework Orange [Zupan et al., 2004]. Simon is described in more details in the Section. 2.3.3.

4.1.2 Exploration algorithm

In the beginning, i.e. at time t_0, the robot has no knowledge about the effects of its actions. For example, it does not know how its actions from the current state influence its

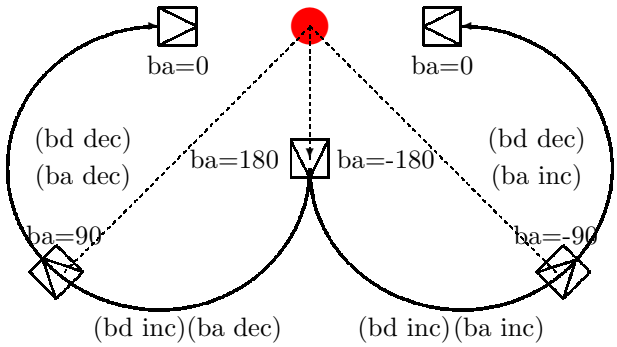


Figure 4.6: The various angles when robot is turning left and right from $ba = 180$ or $ba = -180$. From $ba = 180$ to $ba = 90$ the class value will be $++--$. From $ba = 90$ to $ba = 0$ the class value will be $----$. From $ba = -180$ to $ba = -90$ the class value will be $++++$. From $ba = -90$ to $ba = 0$ the class value will be $--++$.

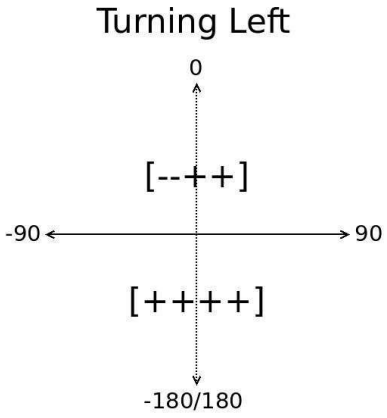


Figure 4.7: The class values for the four quadrants of the angle when the robot is turning left. It is interesting to note here that there are only two class values for the entire space when the robot is turning left.

observations in the next step. Namely, there are no relations known to the robot between actions (L and R) and observations (ba and bd). Without a model the robot can only move by applying random actions, i.e. randomly choose the speed of each wheel. Doing so it collects some data (usually around 10) and learns from this data. The learning is supervised. The attributes are the robot's actions, ratios of the actions and observations. The class variable is defined by qualitative relations (as described in the previous section) between the actions and observations. The robot learns its first model from a small dataset collected by random movement. This initial model is not very accurate and useful. Nevertheless, it enables the robot to use it for making predictions about further actions.

The ability to make predictions enables the choice among various learning modes. A learning mode determines the next action. The most primitive learning mode is random mode, in which the robot chooses one of its three possible actions at random. Random movement is thus defined by actions rather than by robot's positions. The latter is not even possible since in our case the robot is not aware of its coordinates and can not choose to navigate in any coordinate system. The modes used in the algorithm are discussed in detail later in this document.

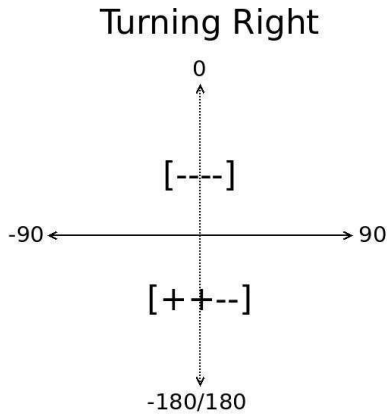


Figure 4.8: The class values for the four quadrants of the angle when the robot is turning right. Similar to turning left, turning right also leads to only two class values for the entire space.

Listing 4.2: The different modes used in the exploration algorithm.

```
1 Random mode
2 Uniform mode
3 Persistent mode
4 NFA mode
```

The robot has to learn the relations between its actions and observations. Qualitative models that the robot is learning are composed of qualitative trees (qtree) and qualitative non-deterministic finite automata (NFA or envisionment) which it builds from the temporal sequence of its actions and observations. There is no relation between the qualitative tree and the envisionment. They are merely a different perspective to the same data. While the qualitative tree is used for prediction, the envisionment is used for planning new experiments for the robot to explore new and less explored regions. Similar to a qualitative tree, the envisionment is gradually learned by the robot.

The robot's exploration algorithm as shown in Listing. 4.3 includes four modes as shown in Listing. 4.2 that strive to guide the learning towards the final goal.

First and most primitive is the *random mode*. While in this mode, the robot moves randomly choosing the actions from its set of available actions. The second mode called *uniform mode* is used when the robot wants to sample the actions so that their distribution is uniform. Uniform distribution of actions assures that the robot is not biased towards one of the actions, e.g. going straight ahead all the time. At first glance it may seem that uniform and random modes are the same, but the difference lies in the fact that uniform mode also accounts for the action executed while in *persistent mode*, which is the third one. The robot, using this mode, keeps executing the same action for some time. Doing so it is collecting more learning examples of the same kind. The last one which is the *NFA mode* is one of the most important mode the robot employs. This mode is used to uniformly sample the class values. Since the robot does not have complete information about how its actions affect the class value, or in other words, how it can know what action to perform in order to reach a particular class value, it uses the NFA.

Fig. 4.9 shows the final NFA learned by the robot. The arcs indicate the actions which cause the transition from one state to the other. The actions are not shown here for rea-

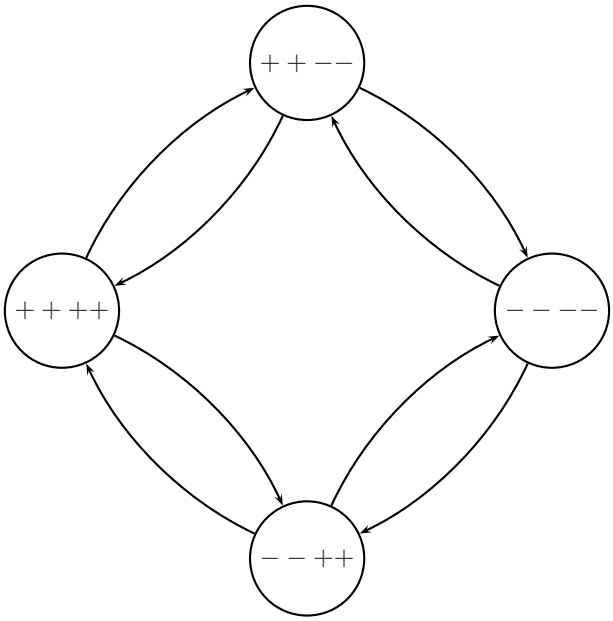


Figure 4.9: The envisionment learned by the robot. It is interesting to note that there are no direct transitions from $++++$ to $----$ or from $++--$ to $--++$ and vice versa. The envisionment alone conveys very little meaning. All it is able to tell the robot is that a particular action from a state will eventually lead it to another state. It does not encapsulate the values of the observation attributes which would trigger this transition. In order to achieve this a combined data structure of both the qualitative tree as well as the envisionment is required.

sons of brevity. One interesting thing to note here is that there are no transitions from $++++$ to $----$ or $++--$ to $--++$. Nevertheless the robot can always reach the state $----$ from $++++$ through either $++--$ or $--++$. Now imagine the robot is in the process of refining its model. While in the NFA mode, it first chooses the class value which has been sampled the least. Then it uses the NFA to find an action that can lead it from its current state to the least sampled state. Thus it is able to sample all the class values uniformly. The changing of modes from one to another based on various environmental and internal conditions prevents the robot from getting stuck in a local minima.

The complete learning algorithm is shown in Listing.refalgorithm. This algorithm only shows the exploration mechanism of the robot. The details on how the various steps like predicting the result of the current action and creating and updating the model are discussed later. The robot uses *random mode* only for its first ten moves when it has no knowledge about its environment and the random choice is the best it can make. After it collects the first ten learning examples it can already build a first model and start using it. At this time, it changes the mode to *uniform* and enters the main loop in which it is updating and improving the model.

The main loop starts with choosing the next action based on the current mode. After the robot picks the action it uses the current qualitative tree to make the prediction using the current state and the action. When it makes the prediction it executes the action and observes the result. It compares its own prediction with the actual observation. In the case of a mismatch, the robot is "surprised" and motivated for further exploration of the unknown behaviors. The reason for the mismatch is the false prediction of qualitative behavior, i.e. the signs in the class value were predicted wrongly. The robot updates the

Listing 4.3: The exploration algorithm used to direct the experiment so that the most significant learning samples get collected in the initial stages of exploration.

```
1  mode = RANDOM
2  perform random movement and after 10 steps build the model, i.e.
       a qualitative tree and an NFA.
3  mode = UNIFORM
4  (try to uniformly sample ACTIONS, not class values! The latter
      is included in the NFA)
5  LOOP define the next action based on mode
6  predict the result of the action with the current qtree
7  execute the action
8  observe the result
9  FT := frequency table of class values
10 compare prediction and the actual observation
11 IF prediction != observation // (i.e. MOTIVATION or SURPRISE)
12          check for changes in qualitative values (class)
13          IF qvalue changed (another MOTIVATION)
14                  add this change (transition) to the NFA
15                  mode = PERSISTENT
16          update the model (qtree)
17          PLANNING (go to the state )
18          in the current state of NFA:
19                  IF the variance between the current class value
                        and the lowest one in FT >10%:
20                          mode = UNIFORM
21                  ELSE
22                          mode = PERSISTENT
23 ELSE
24          mode = PERSISTENT
25          IF prediction correct consistently for a long time
26                  find the lowest frequency class value from FT
27                  mode = NFA
28                  choose a new action and execute until the chose
                        class value is reached
29                  mode = UNIFORM
30 GOTO LOOP
```

NFA with a new state and transition and also updates the qualitative tree. It does not always update the model. Details on how it chooses to update the model is described in Section. 4.1.3. After it updates the model, the robot starts designing a new experiment and planning its actions so that it could carry out the designed experiment. For this purpose it maintains a frequency table of class values and it observes the difference between the number of examples in the current NFA state and the one with the lowest frequency in the table. If the number of examples in the current envisionment state is greater than a threshold, it selects *uniform mode* and picks *persistent mode* otherwise. This finishes one iteration of the loop and starts a new one.

If they match, the robot chooses *persistent mode*. Then it checks if the predictions have been consistently correct for a long time in which case it chooses the least explored class value and switches to *NFA mode*. It then finds from the NFA, an action that lets it reach this class value from the current state. It executes this action until the particular state is reached. After it has reached this state it changes to *uniform mode* in the hope that it will be able to collect learning samples with this low frequency class value to further improve the model.

4.1.3 Creation and Updation of the model

The qualitative models consist of a qtree and the NFA. The qtree is created using the orngTree module from Orange. The module implements a class *TreeLearner* for building both decision and regression trees. TreeLearner is a class that assembles a generic classification tree learner. The classification tree learner is provided with the sample data gathered by the robot. This data has the following attributes.

- L - the derivative of the path of the left wheel which is the left wheel speed
- R - the derivative of the path of the right wheel which is the right wheel speed
- L/R - the ratio of the left and right wheel speeds created by pPadé
- bd - the ball angle
- ba - the ball distance
- C - the class attribute created by pPadé

The classification tree learner returns an Orange TreeClassifier which is the learned qualitative tree. At any time, this classifier may be used to predict the result of an action. Each node of the qtree also holds the training data associated with it. This enables it to classify new test data and predict the class for it. Further details on the Orange classes can be found in Section. 2.3.

Updation or refinement of a model is a slightly more complicated task. Refinement is considered only when the current model provides wrong predictions. Once refinement is taken into consideration, there is the question of whether the refined model can classify better than the previous model. To determine this we use classification accuracy. If the old tree can classify the newly available data 100 percent, there is no need for a refinement. If this is not the case, then the question arises if the new qtree will have an equal or higher classification accuracy with all the data including the newly available one. If the classification accuracy of the new qtree is better, it replaces the old qtree.

Classification accuracy is calculated by using the Area under ROC curve(AUC). In signal detection theory, a receiver operating characteristic (ROC), or simply ROC curve, is a graphical plot of the sensitivity vs. (1 - specificity) for a binary classifier system as its discrimination threshold is varied. [Swets, 1996]. AUC returns the area under ROC

curve (AUC) given a set of experimental results. Its value range from 0 to 1, 1 being the indicator of perfect classification.

4.1.4 Prediction

The prediction is achieved using the qtree built from the learning examples. Prediction of how *bd* and *ba* changes is achieved by providing the qtree with the action L and R the robot is about to execute. The qtree classifies this test data and returns the class value in the classification group. This class value (for eg: $++--$) is used as the prediction of observations for the robot's actions.

Consider an example where the robot is about to take actions $L = 4$ and $R = 5$. These actions along with the constructed attribute L/R is provided as the test data for the qtree. The qtree will classify this and return the class value corresponding to the classification, say $++--$. This is the prediction of the next state. This means that, if the robot executes actions $L = 4$ and $R = 5$, the *bd* will increase and the *ba* will decrease. The action can then be executed by the robot, the results observed and compared with the prediction.

4.2 Experiments and Results

This section describes the experiments conducted and the results obtained to evaluate the work. The experimental setup was created in the simulator Simon bundled with Orange. Simon is a 2D simulator which does not have complex real world physics. In our experiments, this is not required as the robot does not interact with the objects in the world. It is only a passive observer. The L and R are the direct wheel speed commands. *bd* and *ba* are calculated by Simon. L/R is added by pPadé. The robot takes a random initial position which does not coincide with the ball. The robot uses the exploration algorithm to gather relevant learning data in the fewest possible steps. We observed the growth of the qtree learned throughout the process. The model progressively evolves, correcting biases and unbalanced trees. Table. 4.1 shows the learning samples that were collected by the robot in this process after 20 steps. Fig. 4.10 shows the state of the qtree after 20 steps. Fig. 4.11 shows the state of the qtree after 1000 steps. Fig. 4.12 shows the final tree learned after 2674 steps.

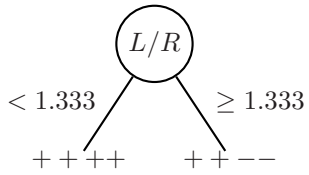


Figure 4.10: The model created by the robot after 20 steps.

The exploration algorithm from the previous section enables the robot to learn by experimentation in an efficient way. To confirm the latter, we compare our approach to the pure random strategy. Again, we stress that random strategy does not mean random sampling of the coordinate space but rather choosing the actions randomly.

In random strategy we use a parameter *duration* which defines the frequency for choosing a random action. If *duration* $= 1$ the action is chosen randomly on each simulation step while for *duration* $= n$ it is chosen only each n-th step and maintained the same in between. The latter is actually not a pure random strategy but rather a mixture of

time	L	R	bd	ba	L/R	Q
1	5.000	5.000	349.137	176.257	1.000	$++++$
2	5.000	5.000	349.536	176.261	1.000	$++++$
3	5.000	5.000	349.935	176.265	1.000	$++++$
4	5.000	5.000	350.334	176.270	1.000	$++++$
5	5.000	5.000	350.733	176.274	1.000	$++++$
6	5.000	5.000	351.133	176.278	1.000	$++++$
7	5.000	5.000	351.532	176.282	1.000	$++++$
8	5.000	5.000	351.931	176.287	1.000	$++++$
9	5.000	5.000	352.330	176.291	1.000	$++++$
10	3.000	5.000	352.650	176.676	0.600	$++++$
11	5.000	3.000	355.847	177.842	1.667	$++--$
12	5.000	3.000	356.167	177.462	1.667	$++--$
13	5.000	3.000	356.167	177.462	1.667	$++--$
14	5.000	3.000	356.486	177.083	1.667	$++--$
15	5.000	3.000	356.806	176.704	1.667	$++--$
16	5.000	3.000	357.125	176.325	1.667	$++--$
17	5.000	3.000	357.444	175.947	1.667	$++--$
18	5.000	3.000	357.763	175.569	1.667	$++--$
19	5.000	3.000	358.082	175.191	1.667	$++--$
20	5.000	3.000	358.401	174.814	1.667	$++--$

Table 4.1: A sub set of the learning samples collected by the robot while learning by observation.

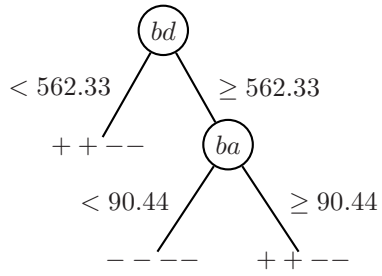


Figure 4.11: The model created by the robot after 1000 steps.

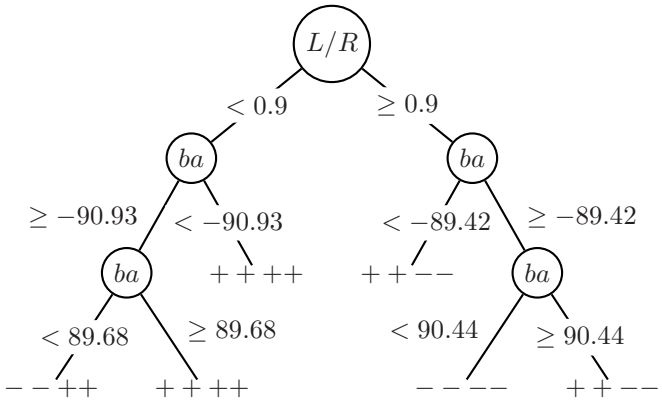


Figure 4.12: The final model created by the robot after 2674 steps.

	Random		our exploration strategy	
Run	Stepsize	Steps to reach best model	Stepsize	Steps to reach best model
1	1	Not until 30000	1	2674
2		Not until 30000	1	3685
3		Not until 30000	1	1991
4	10	Not until 30000	1	2078
5		Not until 30000	1	3530
6		Not until 30000	1	3254
7	100	15967[a]	1	7317
8		Not until 30000	1	4866
9		27654[b]	1	2843

[a]Even this does not result in the ideal model, but very close to it
[b]This resulted in a model separated at the root by L instead of L/R

Table 4.2: Comparison between random action selection and our exploration strategy presented here.

random and persistent. We use it for comparison anyway since the pure random strategy performs extremely poor.

We ran 3 runs of each random strategy, varying *duration* and 9 runs with different initial positions of the robot with our exploration algorithm. We manually determined the point at which the robot learned the desired model. We measured the time it had needed to learn the model in the number of steps it performed until that state. Table 4.2 presents the results over different runs, the averages and standard errors.

The results show that the robot learns significantly better and faster with our exploration algorithm as opposed to the pure random strategy or random-persistent strategies. The reason for devising a complex set of modes is motivated by the way humans experiment. We pursue one direction until there arises a reason or motivation to change it. The reason could be as simple as boredom. It is also clear from the results that a qualitative but sufficient model of the world can be learned in very few steps. This model, is not only a representation of the world, but also a tool for predicting the outcome of the robots actions.

The experiments mentioned above was done using the 2D simulator Simon. Although simulators like Simon without real world physics helps in fast prototyping, the results may sometimes be misleading. In order to verify if our algorithm and representation would work for real robots, we tried to experiment with a simple Lego Mindstorm robot. The experimental scenario and the world was exactly as described above. The ball distance and ball angles were calculated by placing an overhead camera above the experimental area. But due to space constraints we had to limit the area in which the robot could explore. Whenever the robot crossed this limit, it was reset to a similar angle inside the boundaries of the experimental setup. The results we found were enlightening. First of all we found that it takes much more steps for a real robot (around 8000) to learn the qualitative model. This was mainly due to the fact that the distance covered and the angle turned by a real robot is much larger than in the simulation. So it required much more samples so that the space of angles were sampled. Other than that, the same model was learned by the robot.

4.2.1 Feature Selection

The learning of qualitative models using qtrees also ensures a degree of feature selection. In our experiment, we used L, R, ba, bd and L/R as the attributes or features in each of the learning example. But in the final model learned, only ba and L/R are present. These two features were selected from among the five features available. The features L,

R and *bd* were discarded as the robot found that they do not have a direct influence on the output of an experiment. It is interesting to note that the algorithm found L/R to be useful but L and R separately to be useless. This is a clear indication that it can identify hidden high-level actions like turn-left, turn-right and move-straight. It would be rather interesting to try in a future work to learn these higher level actions.

Chapter 5

Learning by experimentation

5.1 Approach

This section outlines the learning of qualitative models by active experimentation with the objects in the world. The task is similar to the one described for learning by observation. The difference is that in this scenario, the robot is allowed to interact actively with the objects thus learning by experimentation. The importance is for the qualitative models learned rather than an exploration algorithm to learn in least number of examples. The learning happens by progressively building qualitative models as and when new information is available. It then uses these models for predicting the result of its actions.

5.1.1 Experimental Domain

The experimental scenario that we explore is known as the movability scenario in the larger project XPERO. In order to avoid ambiguity, we will use the same terminology here. The scenario consists of a robot with 5 boxes as shown in Fig. 5.1. To avoid confusion by us, the movable boxes are colored red and the non-movable boxes are colored blue. This coloring of the boxes were not used by the robot. The robot observes its distance to the box (box distance, labelled as *dist_obj_<object_name>*) and the angle between its orientation and the box (box angle, labelled as *angle_obj_<object_name>*) while driving around pushing the boxes one by one. The box considered here for the measurements is the box which the robot is experimenting or intending to experiment with.

The robot also passively observes the distance and angles to all the other boxes in the environment. This is motivated by the way children experiment with toys. When they are experimenting with a particular toy, they are mostly concerned with that toy alone. Nevertheless they will be attracted to some other toy if something interesting happens. This indicates that they observe the other toys passively. In order to simulate an experimentation like this, we observe the changes in the distances and angles to the other boxes as well. Then we combine all these distances into a single value which indicates if any of them are changing. Similarly we combine the angles. The single value is set to 1 if any of the individual distances or angles are changing. It is set to 0 if none of the individual distances or angles are changing.

The robot tries to experiment and learn which boxes are movable and which are not. There are two possible learned result for movability of boxes. The first one is to learn which box is movable and which is not, based on the label of the box. i.e the robot learns that boxes B and C are not movable and boxes A, D and E are movable. The second way is to learn which box is movable and which is not, based on some feature of the box, say

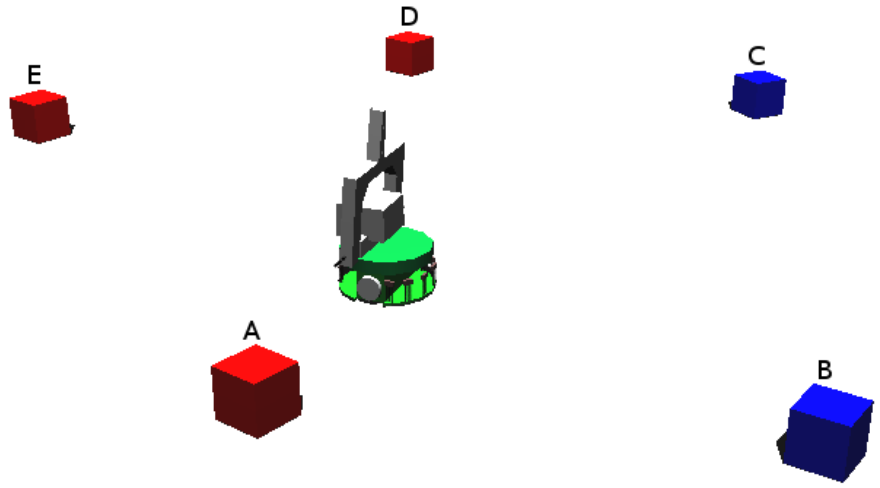


Figure 5.1: The experimental setup for learning by experimentation. The colors of the box are not significant for the experiments. It is only for human understanding. The red boxes are movable and the blue boxes are not movable.

color. The robot eventually learning that red boxes are movable and blue boxes are not movable. Since neither results are perfectly generalisable for all types of boxes, we will look only into one of them here namely the first one. The following features were used for learning in this domain.

- sL' - the left wheel speed
- sR' - the right wheel speed
- *bumper_rob* - the value of bumper sensor(on/off)
- *object* - the name of the object the robot is currently experimenting
- *dist_obj_A* - dist between robot and object named A
- *angle_obj_A* - angle between robot and object named A
- *dist_obj_B* - dist between robot and object named B
- *angle_obj_B* - angle between robot and object named B
- *dist_obj_C* - dist between robot and object named C
- *angle_obj_C* - angle between robot and object named C
- *dist_obj_D* - dist between robot and object named D
- *angle_obj_D* - angle between robot and object named D
- *dist_obj_E* - dist between robot and object named E
- *angle_obj_E* - angle between robot and object named E
- sL'/sR' - ratio of left and right wheel speeds constructed by pPadé

Listing 5.1: The 6-tuple making the class attribute

```
C=<dist_obj_<object_name>L, dist_obj_<object_name>R, angle_obj_<
    object_name>L, angle_obj_<object_name>R, dist_obj_other ,
    angle_obj_other>
```

The class attribute was constructed as follows. The robot has two actions (sL' and sR') and two observation variables (*dist_obj_<object_name>* and *angle_obj_<object_name>*), so it should learn *dist_obj_<object_name>L=Q(s s_L')*, *dist_obj_<object_name>R=Q(s s_R')*, *angle_obj_<object_name>L=Q(s s_L')* and *angle_obj_<object_name>R=Q(s s_R')*. We also concatenate to this the combined behavior of all the other distances *dist_obj_other=B(b [dist_obj_<object1>, dist_obj_<object2>,... , dist_obj_<objectN>])* and all the other angles *angle_obj_other=Q(b [angle_obj_<object1>, angle_obj_<object2>,... , angle_obj_<objectN>])*. Where b is the boolean representing if any of the attributes in the list has changed or not. As mentioned before, *dist_obj_other* is 0 if none of the other distances change and 1 otherwise. *angle_obj_other* is 0 if none of the other angles change and 1 otherwise. We define the class C of this domain as a 6-tuple of signs as shown below.

Thus "$++--11$" means: *dist_obj_<object_name>* is increasing when sL' and sR' are increasing, *angle_obj_<object_name>* is decreasing when sL' and sR' are increasing, all the other distances and angles are changing. "$--++00$" means: *dist_obj_<object_name>* is decreasing when sL' and sR' are increasing, *angle_obj_<object_name>* is increasing when sL' and sR' are increasing, all the other distances and angles are constant.

5.1.2 Learning algorithm

There were several improvements made over the algorithm for learning by observation. One of the main improvement was the ability to experiment in the presence of many objects in the environment. This in turn led to another improvement which was to separate the object under investigation from the rest of the objects in the environment and group all the "other" objects together. A new methodology to label the features were also introduced which led to changes in pPadé to handle for the new labelled data.

The ability to experiment in the presence of many objects in the environment was achieved by adding features for each of the object. Thus new features for distance and angle were added for each of the five objects. Addition of features was not restricted to five objects. New features will be added automatically when new objects are added to the experimental domain. This automatic addition of features was made possible by the definition of interfaces for each attribute like distance, angle etc. Each of these interfaces are responsible for generating features for every object. Thus the interface *distance* will generate the features *dist_obj_A*, *dist_obj_B*, *dist_obj_C*, *dist_obj_D* and *dist_obj_E*. Similarly the interface *angle* is responsible for the generation of the features *angle_obj_A*, *angle_obj_B*, *angle_obj_C*, *angle_obj_D* and *angle_obj_E*. In the future more interfaces like size, color etc may be added based on the availability of sensors or vision algorithms that can detect these features.

The separation of the features of the object under investigation from the rest of the objects is important as the robot at any time is most interested with the object it experiments with. Thus we modified the learning algorithm in such a way as to distinguish the features of the object under experimentation. This was achieved by two things. First, a new attribute called *object* was added to the list of features to identify the object the robot is experimenting. Even though it was added as an attribute to the learning sample,

in reality it acts as both a parameter for pPadé and a learning attribute. The *object* parameter would let pPadé recognize which object is under experimentation by the robot. In our particular experiment, the *object* parameter would take any value from the set A, B, C, D, E which are the names of the objects. Secondly the class attribute was created in a way to give maximum importance to the chosen object. Thus at any time, the first four characters out of six of the class value indicates the change in distance and angle of the object in question with respect to the change in wheel speeds.

The features of the other objects (other than the object in question) is also significant, though it is less significant than the object in question. So the last two characters of the class attribute was dedicated to represent change in distances and change in angles of all the other objects. The first character is 0 if none of the other distances change and 1 otherwise. The second character is 0 if none of the other angles change and 1 otherwise.

Even though we use labelled data for learning, this labelling is not performed by a human. Labelling is done by the appropriate method which is responsible for calculating or measuring a feature. For example, the method which calculates the distance between the robot and an object would automatically generate the measurement under the label *dist_obj_<object_name>*. *dist* indicates that this measurement is of type distance, *obj* indicates is it is a measurement of an object. The other one is *rob* which indicates that it is a measurement of the robot. The *<object_name>* will be the name of the object to which the measurement belongs.

5.1.3 Prediction

It is important to note that the prediction is not as generalized as in the learning by observation scenario. This is due to the fact that the concept of movability is not associated to a particular feature but to a particular object. In real world, movability of an object depends on a lot more features like weight of the object, if the object is fixed to the floor, material and texture of the object which in turn determines the weight of the object etc. It is not a good idea to generalize movability based on attributes like color or size. Size of an object may seem to be a good feature at first, but for example a metal box of 1mx1m may not be movable but a plastic box of 1mx1m may be movable. In this scenario, the best prediction that would be useful is the prediction of whether an object is movable or not based on previous learning experience. When a new object is introduced in the environment, the robot will have to experiment with that object before it can predict if it is movable or not.

5.2 Experiments and Results

This section describes the experiments conducted to evaluate the learning by experimentation scenario described in section 5.1.
The experimental setup was created in the simulator Webots from Cyberbotics. Webots is a 3D simulator which uses the Open Dynamic Engine (ODE) for simulating real world physics. Webots is described more in section 2.4. The sL' and sR' are the direct wheel speed commands. *bumper_rob* is calculated based on proximity. *object* is recorded based on the object the robot is experimenting with. *dist_obj_A*, *angle_obj_A*, *dist_obj_B*, *angle_obj_B*, *dist_obj_C*, *angle_obj_C*, *dist_obj_D*, *angle_obj_D*, *dist_obj_E* and *angle_obj_E*,are calculated by the robot using the Supervisor module of Webots. More about supervisors are described in section 2.4. sL'/sR' is constructed by pPadé using the chain rule. The calculation of sL'/sR' is improved from that of the learning by observation experiment. We observed that in learning by observation, since we used the

numerical value of L/R, there was a slight bias created. This is due to the fact that the qtree is a binary tree. If the robot turning left was chosen as the split for the left subtree, the robot moving straight *and* the robot turning right were together chosen as the split for the right subtree. In order to avoid this, we discretize the ratios calculated by pPadé. The discretization was performed according to the Table. 5.1.

sL'/sR'	
Continuous	Discrete
$sL'/sR' < 0$	$(inf, 0)$
$0 < sL'/sR' < 1$	$(0, 1)$
$sL'/sR' = 1$	(1)
$sL'/sR' > 1$	$(1, inf)$

Table 5.1: The discretization of the ratio of the wheel speeds. sL' and sR' are the speeds of the left and right wheels respectively.

The robot takes a random initial position which does not coincide with any of the objects. The robot then picks an object from the environment and decides to experiment with it. In our experiments, the robot choose objects in alphabetical order of names. It first moves towards object named A. It pushes the object around for a while, until the tree constructed can predict the result of the robots actions. A subset of learning samples collected for object A is as shown in Table. 5.2 and Table. 5.3. The learning samples are split into two tables due to space constraints. A learning sample is formed by appending the rows of both tables. The tree constructed after collecting samples for object A is as shown in Fig. 5.2.

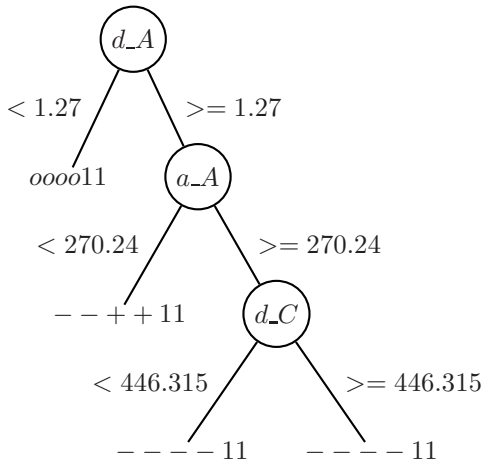


Figure 5.2: The qtree for movability after experimenting with object A. d_A and d_C are the distances from robot to box A and C respectively. a_A is the angle between orientation of robot and box A.

Once the robot is able to consistently predict the outcome of its actions, it moves on to the next object which is B. It tries to push object B, but it does not move at all as it is a non-movable object. Soon the robot builds a model that can correctly predict the outcome of its actions. This model built by the robot after experimenting with both A and B is shown in Fig. 5.3.

Like before, once the robot is able to consistently predict the outcome of its actions, it moves on to object C. It performs the same experiment of pushing as before. As before,

Parameters		*obj_A*		rob				
time	*object*	*dist*	*angle*	*bumper*	sL'	sR'	sL'/sR'	Q
3.000	A	11.578	0.000	off	20.000	20.000	(1)	–oo11
4.000	A	11.103	0.000	off	20.000	20.000	(1)	–oo11
5.000	A	10.624	0.000	off	20.000	20.000	(1)	–oo11
6.000	A	10.142	0.000	off	20.000	20.000	(1)	–oo11
7.000	A	9.660	0.000	off	20.000	20.000	(1)	–oo11
8.000	A	9.182	0.000	off	20.000	20.000	(1)	–oo11
9.000	A	8.709	0.000	off	20.000	20.000	(1)	–oo11
10.000	A	8.238	0.000	off	20.000	20.000	(1)	–oo11
11.000	A	7.764	0.000	off	20.000	20.000	(1)	–oo11
12.000	A	7.287	0.000	off	20.000	20.000	(1)	–oo11
13.000	A	6.807	0.000	off	20.000	20.000	(1)	–oo11
14.000	A	6.330	0.000	off	20.000	20.000	(1)	–oo11
15.000	A	5.857	0.000	off	20.000	20.000	(1)	–oo11
16.000	A	5.389	0.000	off	20.000	20.000	(1)	–oo11
17.000	A	0.000	0.000	on	20.000	20.000	(1)	–oo11
18.000	A	0.000	0.000	on	20.000	20.000	(1)	–oo11
19.000	A	0.000	0.000	on	20.000	20.000	(1)	–oo11
20.000	A	0.000	0.000	on	20.000	20.000	(1)	oo++11
21.000	A	0.000	17.497	on	5.000	20.000	(0,1)	oo++11
22.000	A	0.000	21.794	on	5.000	20.000	(0,1)	oo++11
23.000	A	0.000	22.034	on	5.000	20.000	(0,1)	oooo11
24.000	A	0.000	21.250	on	5.000	20.000	(0,1)	oooo11
25.000	A	0.000	20.921	on	5.000	20.000	(0,1)	oooo11
26.000	A	0.000	19.868	on	5.000	20.000	(0,1)	oooo11
27.000	A	0.000	19.016	on	5.000	20.000	(0,1)	oooo11
28.000	A	0.000	17.745	on	5.000	20.000	(0,1)	oooo11
29.000	A	0.000	16.363	on	5.000	20.000	(0,1)	oooo11
30.000	A	0.000	0.000	on	20.000	20.000	(1)	oooo11
31.000	A	0.000	0.000	on	20.000	20.000	(1)	oooo11
32.000	A	0.000	0.000	on	20.000	20.000	(1)	oooo11
33.000	A	0.000	15.405	on	5.000	20.000	(0,1)	oooo11
34.000	A	0.000	16.385	on	5.000	20.000	(0,1)	oooo11
35.000	A	0.000	0.000	on	20.000	20.000	(1)	oooo11
36.000	A	0.000	0.000	on	20.000	20.000	(1)	oooo11
37.000	A	0.000	0.000	on	0.000	0.000	0	oooo11
38.000	A	0.000	0.000	on	20.000	20.000	(1)	oooo11

Table 5.2: The learning samples consisting of the object attributes of A and the robot. This data is collected by the robot while experimenting with the object A. The attribute Q lists the class value calculated by pPadé. This sample data set is a portion of the learning sample which includes three different class values. The first is $--oo11$ indicating that the robot is moving towards the object keeping its angle steady. The second is $oo++11$ which actually occurs because the box slips from the front of the robot. Nevertheless this particular class value is eliminated from the final tree automatically. This is because of the noise reduction property inherent to the qualitative trees. The last is $oooo11$ indicating that the robot is pushing the object and the object is moving.

the robot builds a model that can correctly predict the outcome of its actions. This model built by the robot after experimenting with both A, B and C is shown in Fig. 5.4.

When the robot is able to consistently predict the outcome of its actions, it moves on

obj_B		*obj_C*		*obj_D*		*obj_E*	
dist	*angle*	*dist*	*angle*	*dist*	*angle*	*dist*	*angle*
25.839	92.366	50.148	179.534	43.300	-105.051	52.275	-144.871
25.863	93.409	50.625	179.527	43.427	-105.670	52.665	-145.182
25.896	94.477	51.106	179.536	43.559	-106.275	53.061	-145.473
25.939	95.554	51.591	179.549	43.697	-106.875	53.461	-145.759
25.990	96.607	52.075	179.544	43.840	-107.491	53.862	-146.059
26.050	97.654	52.556	179.544	43.987	-108.092	54.261	-146.346
26.118	98.702	53.033	179.560	44.137	-108.668	54.659	-146.611
26.194	99.726	53.507	179.561	44.291	-109.253	55.056	-146.886
26.279	100.741	53.985	179.554	44.451	-109.845	55.456	-147.165
26.373	101.779	54.467	179.567	44.617	-110.418	55.862	-147.425
26.476	102.812	54.952	179.576	44.788	-110.994	56.271	-147.685
26.588	103.817	55.436	179.568	44.964	-111.582	56.681	-147.959
26.706	104.818	55.916	179.572	45.143	-112.147	57.088	-148.213
26.832	105.811	56.391	179.587	45.324	-112.693	57.493	-148.452
26.965	106.776	56.867	179.584	45.510	-113.252	57.898	-148.705
27.107	107.736	57.345	179.580	45.700	-113.811	58.307	-148.957
27.267	108.512	57.854	179.339	45.908	-114.636	58.745	-149.457
27.421	109.636	58.322	179.541	46.103	-114.967	59.147	-149.490
27.584	110.612	58.795	179.596	46.304	-115.446	59.554	-149.670
27.756	111.542	59.273	179.606	46.512	-115.971	59.967	-149.894
27.887	109.583	59.636	176.956	46.676	-119.026	60.284	-152.720
28.003	107.593	59.994	174.293	46.858	-122.095	60.607	-155.558
28.104	105.625	60.348	171.670	47.055	-125.120	60.935	-158.353
28.192	103.708	60.693	169.119	47.265	-128.070	61.264	-161.075
28.268	101.797	61.034	166.587	47.488	-130.996	61.597	-163.774
28.333	99.878	61.372	164.056	47.724	-133.920	61.934	-166.471
28.388	97.953	61.707	161.527	47.973	-136.841	62.277	-169.165
28.432	96.028	62.041	159.004	48.237	-139.755	62.626	-171.852
28.464	94.100	62.370	156.485	48.514	-142.662	62.980	-174.533
28.500	95.068	62.812	156.663	48.900	-143.002	63.460	-174.572
28.547	96.023	63.255	156.828	49.286	-143.347	63.940	-174.621
28.603	96.983	63.697	157.000	49.671	-143.677	64.418	-174.660
28.643	95.059	64.026	154.491	49.965	-146.555	64.777	-177.324
28.669	93.123	64.341	151.994	50.262	-149.407	65.128	-179.969
28.697	94.088	64.762	152.207	50.674	-149.666	65.605	-179.954
28.735	95.044	65.184	152.409	51.088	-149.930	66.083	-179.949

Table 5.3: The learning samples consisting of the other object attributes collected by the robot while experimenting with the object A.

to the next object and so on until the last object. Fig. 5.5 shows the final tree learned after 8179 steps. The different models at different time intervals indicate that the robot is capable of evolving its models.

The final qualitative model is divided at the root into two operating regions. The first operating region, depicted by the left sub-tree encapsulates the operating region of the robot while in contact with an object. The second operating region, depicted by the right sub-tree encapsulates the operating region of the robot all the other time. It is interesting to analyse the class values in these operating regions.

First let us analyse the class values in the right sub-tree. $--oo11$ is the only class

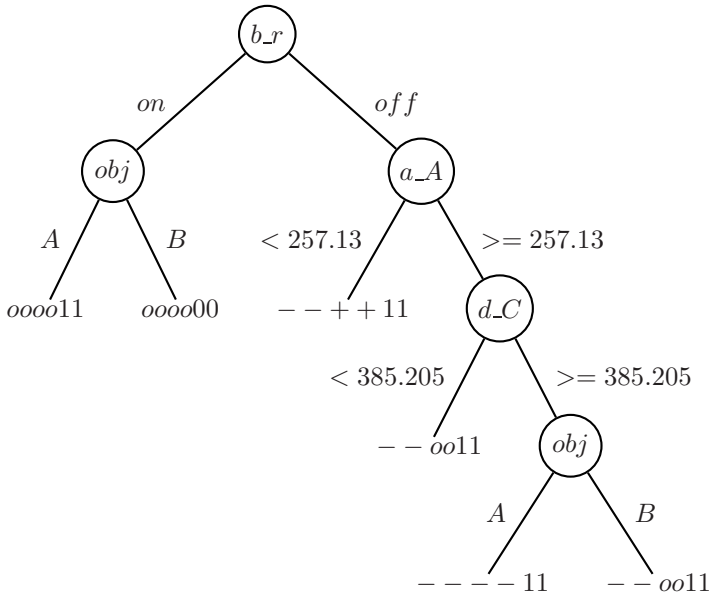


Figure 5.3: The qtree for movability after experimenting with object A and B. *b_r* is the bumper of the robot. *obj* indicates the object under experimentation. *a_A* is the angle between orientation of robot and box A. *d_C* is the distances from robot to box C.

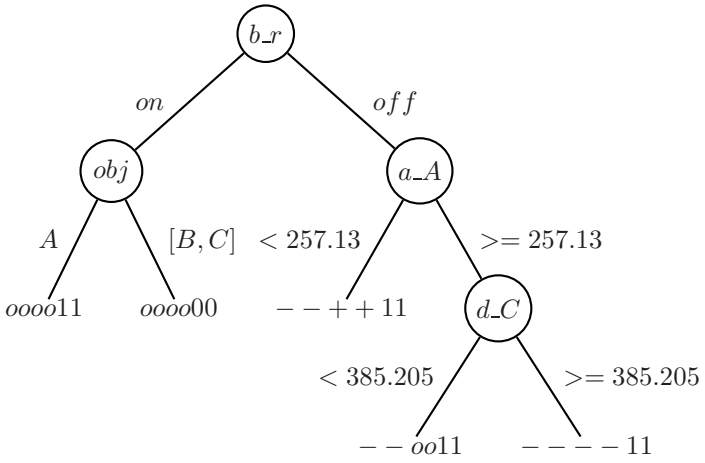


Figure 5.4: The qtree for movability after experimenting with object A, B and C. *b_r* is the bumper of the robot. *obj* indicates the object under experimentation. *a_A* is the angle between orientation of robot and box A. *d_C* is the distances from robot to box C.

value in the right sub-tree. The first two − signs represent the distances w.r.t sL' and sR' and the next two o signs represent the angles w.r.t sL' and sR'. So $--oo$ means that the distance to the box (in all the cases) is decreasing while the angle is steady. The next two characters 11 in the class value represent the state of all the other distances and angles respectively. Thus 11 means that any of the other box distances are changing (increasing or decreasing but not steady) and any of the other box angles are changing. The meaning of this class value in words would be that when the robot is not in contact with any object, it is moving towards an object keeping the angle constant (possibly zero or near zero). This is shown in Fig. 5.6. It is clear from this figure that that the distance to the object decreases and the angle remains constant while the robot approaches the

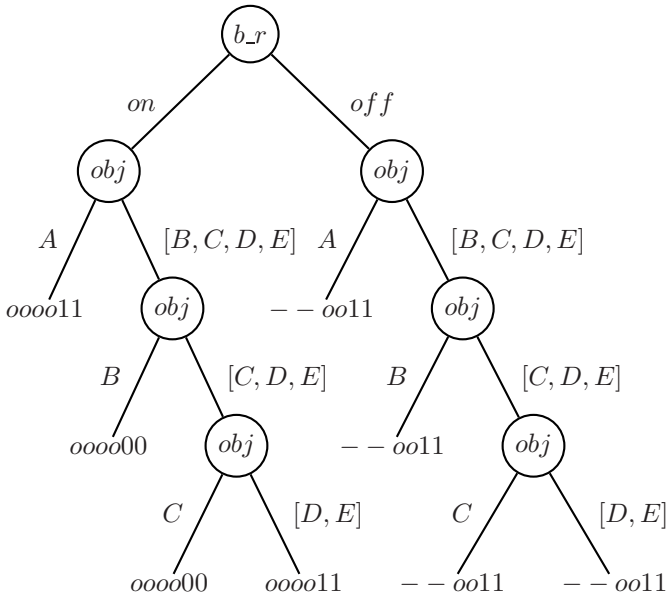


Figure 5.5: The final qtree for movability. *b_r* is the bumper of the robot. *obj* indicates the object under experimentation.

object to experiment. Since movement is relative to the other objects, all or atleast some of the other distances and angles keep changing.

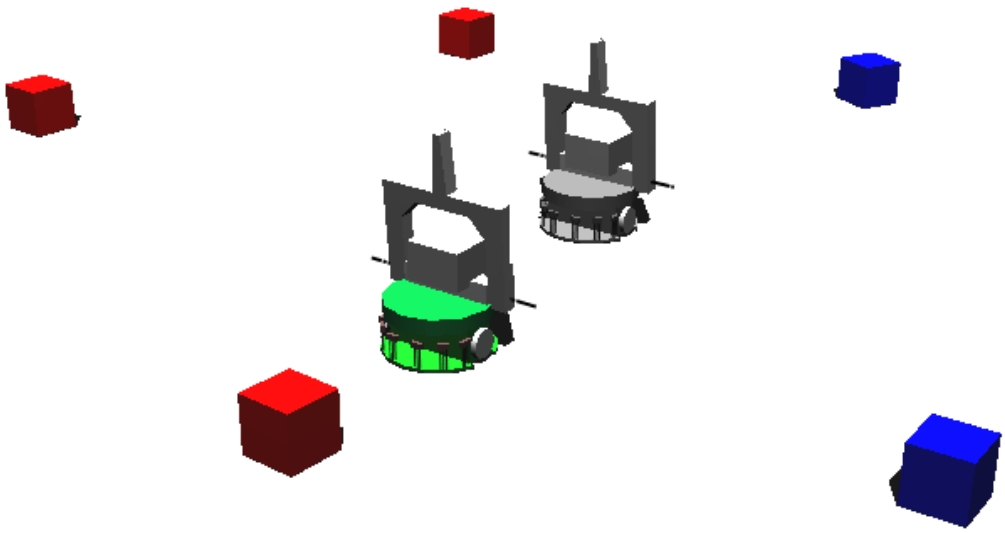


Figure 5.6: The robot approaching the box to experiment. The greyed out robot indicates the initial position of the robot. When it approaches the box A, obviously the distance between itself and box A reduces, but the angle remains almost steady. At the same time its distance and angle to all the other four objects change. This will result in a class value of $--oo11$.

Next we will analyse the class values in the left sub-tree. There are two class values in the right sub-tree, namely *oooo*00 and *oooo*11. The class value *oooo*00 represents the

operating region in which the robot is in contact with a non-movable object. The first four characters of the class value indicates that the robot is in contact (distance is steady) with the object and the angle is also steady. This may not be the case for all the examples for this operating region. Since the robot is pushing against a non-movable object, it may slip sideways thus resulting in the change of the angle. But the interesting thing is that the qualitative tree eliminates these noises. The last two characters of the class value is important in determining if this particular object is movable or not. A 00 indicates that all the other distances and angles remain the same, indicating that neither the robot nor the object is moving. Thus the class value *oooo*00 clearly stands for the non-movable object as shown in Fig. 5.7. Similarly the class value *oooo*11 represents the operating region in which the robot is in contact with a movable object. The first four characters of the class value is same as before, indicating contact with the object. The last two characters of the class value which is 11 here indicates that some or any of the other distances and angles are changing, indicating that both the robot and the object are moving. Thus the class value *oooo*11 clearly stands for a movable object as shown in Fig. 5.8.

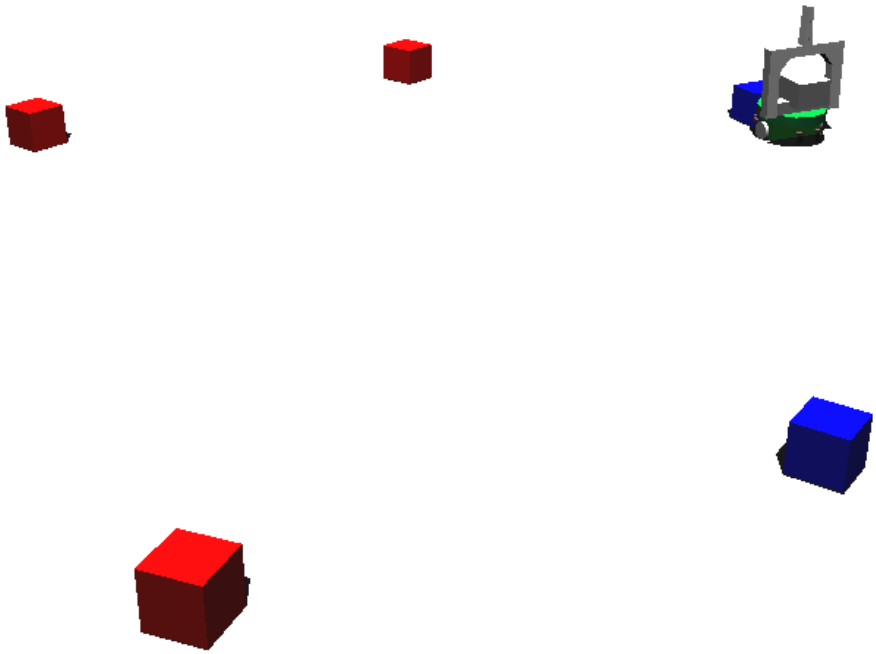


Figure 5.7: The robot pushing box C. Since neither the robot nor the object moves, the distance between itself and box C remains steady, and the angle remains steady as well. At the same time its distance and angle to all the other four objects also remain unchanged. This will result in a class value of *oooo*00.

The splits in the final model is based on the objects because there is no other attribute of the objects or the robot that can characterize whether an object is movable or not. We can deduce from the qualitative tree shown in Fig. 5.5 that objects A,D and E are movable and objects B and C are non-movable.

5.2.1 Feature Selection

Similar to the discussion in the learning by observation, we discuss here about the inherent feature selection of qualitative trees. In our experiment, we used sL', sR', *bumper*, *object*,

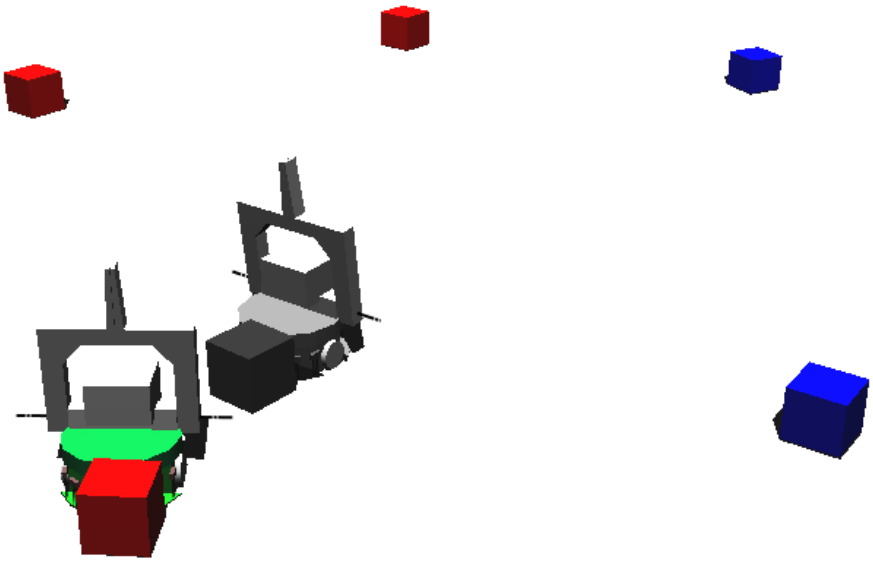


Figure 5.8: The robot pushing box A. The greyed out robot indicates the initial position of the robot. When it pushes the box A, obviously the distance between itself and box A remains unchanged. The angle remains almost steady though it may change slightly. At the same time its distance and angle to all the other four objects change. This will result in a class value of *oooo*11.

dist_obj_A, *angle_obj_A*, *dist_obj_B*, *angle_obj_B*, *dist_obj_C*, *angle_obj_C*, *dist_obj_D*, *angle_obj_D*, *dist_obj_E*, *angle_obj_E* and sL'/sR' as the attributes or features in each of the learning example. But in the final model learned only *bumper* and *object* is present as shown in Fig. 5.5. These two features got selected from among the fifteen features available. The rest of the features were discarded as the robot found that they do not have a direct influence on the output of an experiment. This clearly illustrates the usefulness of qualitative trees for learning. It may even be used for feature selection alone so that another learning system like ILP (Inductive Logic Programming) can then use only those features selected by qtrees. More details on this in Section. 6.1.1

If we were to introduce the color of the objects as part of the attribute list, we will arrive at a tree that will choose the color as a relevant feature. The tree will predict that all red boxes are movable and all blue boxes are not movable. This is a clear indication that no matter what features are used, the tree eventually will choose the most appropriate feature based on the classification accuracy.

Chapter 6

Conclusion

This work proposes a novel qualitative reasoning and representation method for learning by observation and experimentation by an autonomous robot. The experimental results clearly indicate that the proposed method achieved better performance as well as clarity over existing methodologies for the same task.

In Chapter. 4 we proposed a new qualitative representation using classification tree and non-deterministic finite automaton. The class values for classification were computed using pPadé a new method for computing partial derivatives from available data. Unlike previous representations, this representation learns the qualitative landmarks effectively from entirely quantitative data produced by the robot. This is found to be especially useful for robotics where there is abundant amount of quantitative information available from the different sensors at all times. We also proposed a new exploration strategy by which the robot can learn the qualitative model with a very small number of training data. We conducted experiments which were based only on observation. The experiments resulted in excellent qualitative models which were very close to the "ideal model" that can be learned. These models were further used for prediction and directing the experiments in such a way so that the robot can collect the most significant examples first. Table. 4.2 clearly demonstrates that our exploration strategy is much better than simple random sampling of the search space.

In Chapter. 5 we used the same representation and exploration algorithm with few improvements that allowed for generalization. The complexity of the problem domain was increased to see if our method would generalize for larger domains. We went from learning by observation to learning by experimentation. The qualitative representation learned, clearly indicated that our representation and exploration would easily scale up.

We showed a simple example of a robot that is capable of learning by making observations and experiments in its environment. The exploration algorithm that we presented proved to be a useful tool for the autonomous learner that has to design, plan and execute the experiments in order to obtain some knowledge about how its actions influence its observations in the given world. One of the contributions of this work is the learning of qualitative models with quantitative landmarks and the combination of qualitative tree and the envisionment. Both models do not only suffice to support the robot in its actions, but also offer insights into the knowledge that the robot acquired in the learning process. Further, we believe that our approach can be generalized to other more complex domains and that it can scale well due to the simplicity of learning the qualitative models.

We believe that much more has to be done for this approach to be used in entirely different domains. Nevertheless our approach would easily scale up with few changes (if at

all), for domains similar to the ones described here which consists of simple environmental objects like boxes, balls, cylinders, cones etc. It is premature to conclude that this will work in all domains encountered in robotics.The results of the two experiments are a clear indication that it has the potential to be used on much larger and complex domains. It would be very interesting to investigate further in this direction by scaling up the domain. Another interesting work would be to create an evolution of learning. The algorithm for autonomous learning can be further improved by elaborating the planning part and the design of experiments. Applying this procedure in other domains and with real robots may give rise to new ideas for further development. In the following section we describe some of the ideas for future work.

6.1 Future Work

6.1.1 Combining with ILP

Inductive Logic Programming(ILP) is a machine learning method which uses logic programming. ILP systems usually develop predicates from examples and background knowledge. The examples consists of both positive and negative examples. The ILP system will arrive at a hypothesis which entails all the positive examples and none of the negative examples. [Leban et al., 2008] describes an application of an ILP system Hyper [Bratko, 1986] for predicate invention to the same domain described here. This method relies heavily on the specification of the learning problem. The attributes chosen to specify this learning problem is also very important and affects the outcome of the predicate invention. We believe that combining our representation and exploration with this ILP system will greatly reduce this fragility. One way is to use the inherent feature selection mechanism in our method to filter out useless features. Another approach would be to generate the learning samples based on rules that can be learned from the classification tree. These are very coarsely specified approaches which requires much more investigation to arrive at a usable solution.

6.1.2 Learning High-level Actions

Now that we have seen the qualitative model and envisionment the robot learned using the exploration algorithm, let us see how this model can be further used to learn higher-level actions. In the beginning of the learning process, the robot is capable of performing only two actions, namely the direct control of its actuators. In our experiments this is the left and right wheel speeds. These two actions are good enough for the learning scenario considered here. But if we need to generalize this model building technique, we must progressively use the models created in the experiments to enhance the actions of the robot. In other words, the robot should begin building higher level actions from the learned models. Let us see how this translates in our experimental domain.

The robot on completion of the experiment would have learned the qualitative model shown in Figure. 4.12 and the envisionment shown in Figure. 4.9. It knows that there are only two observation attributes, namely *bd* and *ba*. Building an action will be only a matter of defining relations to change these observation attributes using the available actions. In order to do this, a combined data structure of the qualitative tree and the NFA is required. The NFA has the transitions defined by actions between different qualitative states. But it lacks information on the value of *ba* which would make such a transition valid. A combined data structure would have information about both the value of *ba* as well as the action required to reach a particular state. Once this data structure is defined, a high level action such as $go-to-object$ or $go-away-from-object$ may be learned

from this data structure.

For example, a $go - to - object$ action while in a state where $ba = 90$ would translate as an action to turn right. In other words $L/R = (1, inf)$. So the robot can choose L and R in such a manner that $L/R = (1, inf)$. Similarly a $go - away - from - object$ action while in a state where $ba = 90$ would translate as an action to turn left. i.e $L/R = (0, 1)$. These translations can be easily found using the combined data structure of qualitative tree and NFA. Further work is required to include steady states for the qualitative values. For example, in the above mentioned example, the robot will turn right from $ba = 90$ for $go - to - object$, but when it crosses $ba = 0$, it will turn left for $go - to - object$ and start oscillating across $ba = 0$ with alternating actions because it does not know how to keep $ba = 0$ and steady. With steady states introduced, can easily turn right from $ba = 90$ until it reaches $ba = 0$ and then go straight to maintain a steady state of ba. This way it can $go - to - object$ in an intelligent manner.

Learning high-level actions are definitely challenging, but it will be worth the effort spent as it provides infinite possibilities for the robot to learn. Learning can continue in an evolutionary manner even for very complex scenarios. With every new action learned, the robot can perform more complex experiments and build new actions from the models learned. It will be both a challenging as well as exciting work to investigate in this direction.

Bibliography

[Barto et al., 2004] Barto, A. G., Singh, S., and Chentanez, N. (2004). Intrinsically motivated learning of hierarchical collections of skills. *International Conference on Developmental Learning*.

[Berleant and Kuipers, 1992] Berleant, D. and Kuipers, B. (1992). Qualitative-numeric simulation with q3. *In Boi Faltings and Peter Struss (Eds.), Recent Advances in Qualitative Physics, MIT Press*.

[Berleant and Kuipers, 1997] Berleant, D. and Kuipers, B. (1997). Qualitative and quantitative simulation: bridging the gap. *Artificial Intelligence*, 95(2):215–255.

[Berleant, 1991] Berleant, J. D. (1991). *The use of partial quantitative knowledge with qualitative reasoning*. PhD thesis, University of Texas at Austin, Artificial Intelligence Laboratory, Technical Report AI 91-163. (Doctoral dissertation, Department of Computer Sciences.).

[Bradley and Stolle, 1996] Bradley, E. and Stolle, R. (1996). Automatic construction of accurate models of physical systems. *Annals of Mathematics and Artificial Intelligence*, 17(1-2):1–28.

[Bratko, 1986] Bratko, I. (1986). *Prolog programming for artificial intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Bratko et al., 1992] Bratko, I., Muggleton, S., and Varsek, A. (1992). Learning qualitative models of dynamic systems. *in S. Muggleton editor. Inductive Logic Programming, Academic Press, London.*

[Bratko and Suc, 2004] Bratko, I. and Suc, D. (2004). Learning qualitative models. *AI Mag.*, 24(4):107–119.

[Capelo et al., 1996] Capelo, A. C., Ironi, L., and Tentoni, S. (1996). The need for qualitative reasoning in automated modeling: a case study. *In Proceedings of the Tenth International Workshop on Qualitative Reasoning QR'96*, pages 32–39.

[Cohn and Hazarika, 2001] Cohn, A. and Hazarika, S. (2001). Qualitative spatial representation and reasoning: An overview. *Fundamenta Informaticae*, 46:1–29.

[Cohn, 1997] Cohn, A. G. (1997). Qualitative spatial representation and reasoning techniques. In *KI '97: Proceedings of the 21st Annual German Conference on Artificial Intelligence*, pages 1–30, London, UK. Springer-Verlag.

[Cohn et al., 1997] Cohn, A. G., Bennett, B., Gooday, J., and Gotts, M. (1997). Qualitative spatial representation and reasoning with the region connection calculus. *GeoInformatica*.

[Coiera, 1989] Coiera, E. W. (1989). Generating qualitative models from example behaviours. Technical report, Technical Report 8901, University of New South Wales, Deptartment of Computer Science, May.

[Collins and Forbus, 1987] Collins, J. and Forbus, K. (1987). Reasoning about fluids via molecular collections. *Proceedings of the American Association for Artificial Intelligence (AAAI-87), Seattle, Washington*, pages 590–594.

[Crawford et al., 1990] Crawford, J., Farquhar, A., and Kuipers, B. (1990). QPC: a compiler from physical models into qualitative differential equations. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 365–372, Menlo Park, California. AAAI Press.

[Daniel J. Clancy and Kay, 1997] Daniel J. Clancy, G. B. and Kay, H. (1997). Model revision: techniques and tools for analyzing simulation results and revising qualitative models. *Working notes from the 11th International Workshop on Qualitative Reasoning about Physical Systems (QR-97).*

[de Kleer, 1990] de Kleer, J. (1990). Multiple representations of knowledge in a mechanics problem-solver. *Readings in qualitative reasoning about physical systems*, pages 40–45.

[de Kleer and Brown, 1984] de Kleer, J. and Brown, J. (1984). A qualitative physics based on confluences. *Artificial Intelligence 24*, 1-3:7–83.

[Demsar and Zupan,] Demsar, J. and Zupan, B. *Orange: From Experimental Machine Learning to Interactive Data Mining.* AI Laboratory at Faculty of Computer and Information Science, University of Ljubljana.

[Dvorak and Kuipers, 1989] Dvorak, D. and Kuipers, B. (1989). Model-based monitoring of dynamic systems. In *Proc. 11th Int. Joint Conf. on Artificial Intelligence (IJCAI-89)*, pages 1238–1243, San Mateo, CA. Morgan Kaufmann.

[Dvorak and Kuipers, 1991] Dvorak, D. and Kuipers, B. (1991). Process monitoring and diagnosis: A model-based approach. *IEEE Expert*, 6(3):67–74.

[Dvorak, 1992] Dvorak, D. L. (1992). *Monitoring and diagnosis of continuous dynamic systems using semiquantitative simulation.* PhD thesis, University of Texas at Austin, Austin, TX, USA.

[Dzeroski and Todorovski, 1995] Dzeroski, S. and Todorovski, L. (1995). Discovering dynamics: From inductive logic programming to machine discovery. *Journal of Intelligent Information Systems*, 4(1):89–108.

[Falkenhainer, 1990] Falkenhainer, B. (1990). *A unified approach to explanation and theory formation*, chapter 6. Morgan Kaufman.

[Farquhar, 1993] Farquhar, A. (1993). Automated modeling of physical systems in the presence of incompleteknowledge. Technical Report AI93-207, Department of Computer Sciences, the University of Texas at Austin.

[Farquhar, 1994] Farquhar, A. (1994). A qualitative physics compiler. *National Conference on Artificial Intelligence*, pages 1168–1174.

[Farquhar and Brajnik, 1994] Farquhar, A. and Brajnik, G. (1994). A semi-quantitative physics compiler. *In Working Papers of the International Workshop on Qualitative Reasoning (QR-94).*

[Forbus, 1980] Forbus, K. (1980). Spatial and qualitative aspects of reasoning about motion. *Proceedings of the first annual conference of the American Association for Artificial Intelligence, August.*

[Forbus, 1983] Forbus, K. (1983). Measurement interpretation in qualitative process theory. *IJCAI-83.*

[Forbus, 1988] Forbus, K. (1988). Qpe: Using assumption-based truth maintenance for qualitative simulation. *The International Journal for Artificial Intelligence in Engineering*, 3(4):200–215.

[Forbus, 1989] Forbus, K. (1989). Introducing actions into qualitative simulation. *Proceedings of the 11th International Joint Conference on Artificial Intelligence*.

[Forbus, 1995] Forbus, K. (1995). Qualitative spatial reasoning: Framework and frontiers. *In: Glasgow, J ., Narayanan, H ., & Chandrasekaren, B. (Eds.), Diagrammatic Reasoning: Computational and Cognitive Perspectives, AAAI Press*.

[Forbus, 1996] Forbus, K. (1996). Qualitative reasoning. *CRC Hand-book of Computer Science and Engineering. CRC Press.*

[Forbus et al., 1991] Forbus, K., Nielsen, P., and Faltings, B. (1991). Qualitative spatial reasoning: The clock project. *Artificial Intelligence*, 51(1-3):417–471.

[Forbus et al., 1990] Forbus, K. D., Nielsen, P., and Faltings, B. (1990). Qualitative kinematics: a framework. *Readings in qualitative reasoning about physical systems*, pages 559–567.

[Gerceker and Say, 2006] Gerceker, R. K. and Say, A. C. (2006). Using polynomial approximations to discover qualitative models. Technical report, Darthmouth College, Hanover, New Hampshire, USA.

[Hart et al., 2005] Hart, S., Grupen, R., and Jensen, D. (2005). A relational representation for procedural task knowledge. In *Proc. 20th National Conf. on Artificial Intelligence*.

[Hernandez et al., 1995] Hernandez, D., Clementini, E., and Felice, P. D. (1995). Qualitative distances. In W Kuhn A Frank, editor, Spatial Information Theory: a theoretical basis for GIS, number 988 in LNCS.

[Kay, 1997] Kay, H. (1997). Robust identification using semiquantitative methods. *IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS'97), Hull, UK*, pages 26–28.

[Kay, 1998] Kay, H. (1998). Sqsim: a simulator for imprecise ode models. *Computers and Chemical Engineering*, 23(1):27–46.

[Kay et al., 2000] Kay, H., Rinner, B., and Kuipers, B. (2000). Semi-quantitative system identification. *Artif. Intell.*, 119(1-2):103–140.

[Kuipers, 1996] Kuipers, B. (1996). A hierarchy of qualitative representations for space. *Lecture Notes in Computer Science*, 1404:337.

[Kuipers et al., 2006] Kuipers, B., Beeson, P., Modayil, J., and Provost, J. (2006). Bootstrap learning of foundational representations. *Connection Science*, 18(2):145–158.

[Kuipers and Berleant, 1988] Kuipers, B. and Berleant, D. (1988). Using incomplete quantitative knowledge in qualitative reasoning. *National Conference on Artificial Intelligence*.

[Kuipers, 1984] Kuipers, B. J. (1984). Commonsense reasoning about causality: deriving behavior from structure. *Artificial Intelligence*, 24:169–203.

[Kuipers, 1993a] Kuipers, B. J. (1993a). Reasoning with qualitative models. *Artificial Intelligence 59:*, 59:125–132.

[Kuipers, 1994] Kuipers, B. J. (1994). *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge.* Cambridge, MA: . MIT Press.

[Kuipers, 1993b] Kuipers, H. K. . B. (1993b). Numerical behavior envelopes for qualitative simulation. *Proceedings of the National Conference on Artificial Intelligence (AAAI-93), AAAI/MIT Press*.

[Leban et al., 2008] Leban, G., Žabkar, J., and Bratko, I. (2008). An experiment in robot discovery with ilp. *In Proceedings of the 18th International Conference on Inductive Logic Programming*.

[Liu, 1998] Liu, J. (1998). A method of spatial reasoning based on qualitative trigonometry. *Artificial Intelligence*, 98(1-2):137–168.

[Ljung, 1986] Ljung, L. (1986). *System identification: theory for the user*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Michel, 2004] Michel, O. (2004). Webots: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42.

[Modayil and Kuipers, 2007] Modayil, J. and Kuipers, B. (2007). Where do actions come from? autonomous robot learning of objects and actions. *AAAI Spring Symposium Series 2007, Control Mechanisms for Spatial Knowledge Processing in Cognitive / Intelligent Systems*.

[Mohan, 2008] Mohan, A. (2008). Model building and identication for computational models of surprise. Technical Report TR-BRS-CS-2008-03, University of Applied Science Bonn-Rhein-Sieg, Sankt Augustin, Germany.

[Mohan et al., 2007] Mohan, A., Juarez, A., and Henne, T. (2007). A qualitative prediction-observation loop for learning by experimentation. In *Eurosim*.

[Mozetic, 1987] Mozetic, I. (1987). Learning of qualitative models. In *EWSL*, pages 201–217.

[Mugan and Kuipers, 2008] Mugan, J. and Kuipers, B. (2008). Continuous-domain reinforcement learning using a learned qualitative state representation. *QR'08 Workshop on Qualitative Reasoning*.

[Muggleton and Buntine, 1988] Muggleton, S. and Buntine, W. (1988). Machine invention of first order predicates by inverting resolution. In *ML88*, pages 339–351. MK.

[Muggleton and Feng, 1990] Muggleton, S. and Feng, C. (1990). Efficient induction of logic programs. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 368–381. Ohmsma, Tokyo, Japan.

[Nielsen, 1988] Nielsen, P. (1988). A qualitative approach to rigid body mechanics. Technical report, Urbana, Illinois: University of Illinois at Urbana-Champaign, Department of Computer Science.

[Rajagopalan, 1994] Rajagopalan, R. (1994). A model for integrated qualitative spatial and dynamic reasoning about physical systems. In *National Conference on Artificial Intelligence*, pages 1411–1417.

[Richards et al., 1992] Richards, B. L., Kraan, I., and Kuipers, B. J. (1992). Automatic abduction of qualitative models. *Proceedings of the National Conference on Artificial Intelligence (AAAI-92), AAAI/MIT Press*.

[Say and Kuru, 1996] Say, A. C. C. and Kuru, S. (1996). Qualitative system identification: deriving structure from behavior. *Artif. Intell.*, 83(1):75–141.

[Shen and Leitch, 1991] Shen, Q. and Leitch, R. (1991). Synchronized qualitative simulation in diagnosis. *In Working Papers from the Fifth International Workshop on Qualitative Reasoning about Physical Systems*, pages 171–185.

[Stoytchev, 2005] Stoytchev, A. (2005). Behavior-grounded representation of tool affordances. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Suc and Bratko, 2001] Suc, D. and Bratko, I. (2001). Induction of qualitative trees. In *EMCL '01: Proceedings of the 12th European Conference on Machine Learning*, pages 442–453, London, UK. Springer-Verlag.

[Swets, 1996] Swets, J. (1996). *Signal detection theory and ROC analysis in psychology and diagnostics : collected papers*.

[Ungar, 1993] Ungar, H. K. . L. H. (1993). Deriving monotonic function envelopes from observations. *In Working Papers of the Seventh International Workshop on Qualitative Reasoning about Physical Systems (QR'93), Orcas Island, Washington*.

[UvA, 2005] UvA (2005). *User manual for single user workbench*. UvA.

[Webots,] Webots. http://www.cyberbotics.com. Commercial Mobile Robot Simulation Software.

[Weld and de Kleer, 1990] Weld, D. S. and de Kleer, J. (1990). *Readings in qualitative reasoning about physical systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Zabkar et al., 2007] Zabkar, J., Bratko, I., and Demsar, J. (2007). Learning qualitative models through partial derivatives by pade. *QR*.

[Zimmermann, 1993] Zimmermann, K. (1993). Enhancing qualitative spatial reasoning - combining orientation and distance. *In Proc. International Conference on Spatial Information Theory. A Theoretical Basis for GIS, Elba, Italy,*, pages 69–76.

[Zupan and Demsar, 2004] Zupan, B. and Demsar, G. L. J. (2004). Orange: Widgets and visual programming, a white paper. Faculty of Computer and Information Science, Ljubljana, Slovenia.

[Zupan et al., 2004] Zupan, B., Leban, G., and Demšar, J. (2004). Orange: Widgets and visual programming, a white paper.